

ElectricCommander 5.0.4

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

Copyright © 2002 – 2015 Electric Cloud, Inc. All rights reserved.

Published 2/11/2015

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

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

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

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

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

Trademarks

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

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

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

Contents

Welcome to ElectricCommander	1
Overview	1
What makes ElectricCommander unique?	2
What you do	2
What Commander does	2
Schedules	4
Workflows	4
Projects	4
Procedures and Steps	4
Jobs	5
Resources	5
Commander architecture	5
Simple architectural overview	6
Expanded remote configurations	6
Getting Started	8
Overview	8
Basic Commander terminology	8
Getting Started Scenarios help topic series	10
Navigating the Commander user interface	10
The Home tab	11
Projects, Jobs, and Workflow tabs	13
Cloud tab	14
Artifacts and Search tabs	16
Search tab	16
Administration tab	16
Getting Started - Scenario 1 - Creating a simple procedure	18
Overview	19
Begin Scenario 1	19
Scenario extension - Adding another step	25
Finding a Commander web page...	26
Summary	27

Getting Started - Scenario 2 - Creating a procedure that uses an SCM	28
Overview	29
Begin Scenario 2	30
Scenario extension - Add a step to show workspace contents	38
Summary	39
Getting Started - Scenario 3 - Notification, scheduling, and reporting	40
Overview	41
Begin Scenario 3	41
Scenario extension - Add a thumbnail report to your Home page	52
Summary	53
Getting Started - Scenario 4 - Multi-agent build and test	54
Overview	55
Begin Scenario 4	55
Scenario extension - Using postp	61
Summary	63
Configuring ElectricCommander	64
The table below provides a brief description of Commander configuration tasks	64
Web Interface Online Help System	65
Setting Up Resources	66
What is a resource?	66
Commander supports two resource types	66
Standard Resources	66
Proxy Resources	66
Setting up SSH keys	67
To create a new resource	67
Gateways and zones	68
What is a zone?	68
Cross-zone communication	68
What is a gateway?	69
Support for gateways	69
Setting Up Workspaces	70
To create a new workspace	70
Setting Up Email Configurations	71
To create an email configuration	71
Setting Up a Source Control Configuration	72
Setting Up Directory Providers	73
To specify a directory provider	73

Enable/Disable Local Commander Users	73
wrapper.conf Properties	73
The Home Page	74
Overview	74
Job Configurations	74
Create Job Configurations three ways	74
Shortcuts	75
Create Shortcuts two ways	75
Jobs Quick View	75
Create a job category	75
Reports	76
Using ElectricCommander in Your Environment	77
What's in a step?	77
Where's the script for a step?	78
Environment variables	79
When do you use subprocedures?	79
How do you evolve procedures?	79
Porting existing scripts	80
Commander project version control	81
Using ectool and the Commander API	82
Using ectool	82
Logging in	82
Global Arguments (optional)	83
Passing Lists as Arguments	83
Using Perl	83
Perl API structure	84
Common Global Options	86
The Batch API	89
Using the Batch API	89
Installing Commander Perl modules into Your Perl Distribution	91
Installing Perl Modules into the Commander Perl Distribution	91
When Upgrading Commander	92
API Commands - listed by group and in alphabetical order	93
API commands listed by group	93
ACL Management (access control list)	93
Artifact Management	94
Credential Management	96
Database Configuration	96

Directory Provider Management	96
Email Configuration and Management	97
Email Notifiers Management	97
Gateway/Zone Management	98
Job Management	98
Parameter Management	100
Plugin Management	101
Procedure Management	101
Project Management	102
Property Management	102
Resource Management	103
Schedule Management	104
Server Management	104
User/Group Management	105
Workflow Definition Management	106
Workflow Management	107
Workspace Management	107
Miscellaneous	108
API commands listed in alphabetical order	109
API commands - ACL Management	122
API commands - Applications	147
API commands - Application Tier	152
API commands - Artifact Management	158
API commands - Component	188
API Commands - Credential Management	197
API Commands - Database Configuration	206
API Commands - Directory Provider Management	209
API Commands - Email Configuration Management	222
API Commands - Email Notifier Management	228
API Commands - Environment Requests	241
createEnvironment	241
createEnvironmentInventoryItem	242
deleteEnvironment	244
deleteEnvironmentInventoryItem	245
getEnvironment	246
getEnvironments	246

getEnvironmentApplications	247
getEnvironmentInventory	248
getEnvironmentInventoryItem	249
getEnvironmentInventoryItems	250
modifyEnvironment	251
modifyEnvironmentInventoryItem	252
API Commands - Environment Tier	254
createEnvironmentTier	254
deleteEnvironmentTier	255
getEnvironmentTier	256
getEnvironmentTiers	257
modifyEnvironmentTier	257
API Commands - Gateways/Zones Management	259
API Commands - Job Management	269
External Job APIs	286
API Commands - Parameter Management	303
API Commands - Plugin Management	323
API Commands - Procedure Management	329
API Commands - Process	350
createProcess	350
deleteProcess	351
getProcess	353
getProcesses	354
modifyProcess	355
runProcess	356
API Commands - Process Dependency	359
createProcessDependency	359
deleteProcessDependency	360
getProcessDependencies	362
modifyProcessDependency	363
API Commands - Process Step	365
createProcessStep	365
deleteProcessStep	368
getProcessStep	369
getProcessSteps	370
modifyProcessStep	371
API Commands - Project Management	375

API Commands - Property Management	380
API Commands - Resource Management	416
API Commands - Schedule Management	436
API Commands - Server Management	444
API Commands - Tier Map	453
createTierMap	453
deleteTierMap	454
deleteTierMapping	455
getTierMaps	456
modifyTierMap	457
API Commands - User/Group Management	459
API Commands - Workflow Management	472
API Commands - Workflow Definition Management	481
API Commands - Workspace Management	500
API Commands - Miscellaneous Management	507
API Response and Element Glossary	532
Element Glossary	555
Access Control	590
Overview	590
Privileges	590
Users and Groups	591
Special Users and Groups	591
Access Control Lists (ACLs) - allow and deny	591
Inheritance	592
System Objects	593
Access Control and Jobs	594
Examples for Increased Security	595
Example 1 - Basic ACL Setup	595
Example 2 - Team ACL Setup	599
Optional: Restricting Resources and Workspaces by Team	605
Artifact Management	607
Overview	607
Artifact objects	607
Artifact	608
Artifact version	608
Repository	611

Access control	614
Publishing artifact versions	616
Enable compression	616
Include / Exclude patterns	617
Retrieving artifact versions	617
Supplying filters during a retrieve operation	619
Dependent artifact version(s)	620
Artifact cache	621
Cleaning up repositories and caches	623
Authenticating Users for LDAP and Active Directory	624
Configuring LDAP	624
Determining LDAP Mapping	626
Sample LDAP User Record	626
Sample LDAP Group Record	627
Sample Active Directory Configuration File	627
Credentials and User Impersonation	629
What is a credential?	629
Defining a credential	629
Why use credentials?	629
Credential access control	630
Using credentials for impersonation	630
Setting the impersonation credential	631
Accessing credentials from a step	632
Attaching a credential to a step	632
Passing credentials as parameters	632
Credential references	633
Best Practices for retrieving credentials	633
Customizing the Commander UI	635
Customizing parameters	635
How do you customize parameters?	635
Custom parameter form contents	635
Customizing the tab layout	637
Overview	637
View definition syntax	637
Storing views	640
Default views	640
Developing and troubleshooting	640
Home page configuration	641

User settings	641
Location	641
Installing the Home page	641
Reconfigure Contact Support link	641
Defect Tracking	643
Example: Enabling the JIRA integration in your procedure	643
Defect Tracking	646
Example: Enabling the JIRA integration in your procedure	646
ElectricSentry	649
How ElectricSentry works	649
Configuring ElectricSentry	649
Quiet time	649
Resource	650
Polling frequency	650
Time-of-day and day-of-week	650
Configuring a build for continuous integration	650
Source Control Management (SCM) configurations	651
Create a procedure	651
Create a schedule	651
Optional - running ElectricSentry on multiple resources	651
Overview	652
Configuring ElectricSentry to use multiple resources	652
The Job Step Execution Environment	654
Terms and definitions	654
Preflight Builds	657
Why use Preflight Builds?	657
Preflight Build Solution	657
Preflights with Commander	657
Workflow	657
Components	658
Installation	658
Configuration	658
Samples	660
Default SCMs	665
Perforce	665
Subversion	666
AccuRev	667

ClearCase	667
Bazaar	668
Git	668
CVS	669
Mercurial	669
Repo	669
StarTeam	670
TFS	670
Vault	670
Troubleshooting	671
Client	671
Agent	671
Properties	672
Overview	672
Creating or modifying properties	673
Using property values	674
Property sheets and intrinsic properties	674
Property names and paths	675
Absolute property paths	675
Relative property paths	677
Context-relative "shortcuts" to property paths	678
Property path shortcuts in ElectricCommander 5.0 and later	681
Property name substitutions	683
Expandable properties	683
Custom property names and values	683
The property hierarchy	684
Special property references	685
increment	685
timestamp	685
javascript	686
Commander Intrinsic Properties listed by object type	687
Commander Property Type definitions	688
Commander Object / Property tables	689
Property error codes	751
Postprocessors: Collecting Data for Reports	754
Overview	754
Postp	755
Extending postp: matchers	755

Postp functions	756
Integration with the Commander user interface	758
Custom property names and values	758
Diagnostic information	759
Postp integration with Java Tools	760
The process... ..	760
Java Tool matcher examples	761
Artifacts directory	762
Reports	763
Run reports on a non-local resource	763
Real-time reports	763
Home page	763
Jobs	763
Job details / Step details	764
Resources	764
Build reports using ecreport	764
Example 1	766
Example 2	766
Default Batch reports (Run Reports)	768
Report types	768
Default Batch Reports	768
Tip... ..	769
Advanced reporting information - ecrptdata	770
How are default batch reports generated?	770
Filtering	772
Adding additional data columns to the report	772
Secure login	773
Optional Batch reports	774
Creating a report job	774
Viewing the report	776
Optional Batch report examples	776
Category sample report	776
Procedure Usage sample report	778
Count Over Time sample report	779
Multiple Series report	782
Optional Batch reports	786
Creating a report job	786
Viewing the report	788

Optional Batch report examples	788
Category sample report	788
Procedure Usage sample report	790
Count Over Time sample report	791
Multiple Series report	794
Creating Custom reports	798
Data extraction	799
Using ecextract.pl for data extraction	799
Examples	802
BIRT Report Designer	805
Understanding additional report components	806
Packaging reports for Commander	806
Helper functions in ReportUtils.pm That are Part of the ElectricCommander Perl Module	809
Custom Report Examples	811
Example 1: modifying an existing report - adding a "banner" heading	811
To copy an existing Commander report	811
To modify the copied report design	813
Test your new report	815
Example 2: complete end-to-end example	816
Planning this report example	816
Creating a new BIRT rptdesign file	817
Deploy your custom report	827
Test your new report	828
Workflow Overview	831
Workflow objects	832
Workflow Definition	832
State Definition	833
Transition Definition	833
Workflow	834
State	834
Transition	834
Access Control	835
Property Search Paths	835
Parameters	836
Sending Notifications	836
Workflow Logs	836
Visualizing a Workflow	836
Building a Workflow - a Tutorial	837

To begin...Calling a job from within a workflow	838
Collecting a parameter when a workflow is launched and passing its value to a job	840
Retrying a state	842
Automatically transitioning to different states based on the outcome of a job	844
Invoking another workflow	847
Running jobs in parallel	848
Automatically transitioning to different states based on the outcome of jobs in another workflow	849
Waiting for manual intervention	852
Restricting who can take a manual transition	856
Sending email notifiers	859
Adding a global parameter to use later in the workflow	861
Setting the name of your workflow	863
Workflow List	865
Another workflow example	867
Workspaces and Disk Space Management	868
Overview	868
Defining Workspaces	868
Using Workspaces	869
Workspace Directory Names	870
Working Directories	870
Workspace Accessibility	870
Local Workspaces (Disconnected Workspaces)	871
Access control	872
Impersonation and workspaces	872
Commander managed files	873
Disk space management	873
Properties Changed to UUIDs	873
Reasons for the Change	874
Impact for Commander 4.2 Users	874
Actions to Take	876
Migration Tool	876
Default Job Name and Workflow Name Template Changes	877
Disadvantages of Using jobId and workflowId	877
Using the Migration Tool	878
Advantages of using jobCounters	878
What the Script Does	878
Running the Migration Tool	879
jobCounter Considerations	880

Job Name Template and Workflow Name Template Best Practices	880
Actions that Will Affect the Counter	881
Commander 5.0 Name Template Defaults	881
Commander Installed Tools	883
Examples	886
Examples	891
ecproxy Algorithm	894
ecproxy Operations	894
Available Helper Functions	898
Examples	901
"Real World" Examples	902
ClusterExec	902
MySQL	902
Android	903
Copying Other Files from the Workspace	906
Web Interface Help	909
Access Control	910
Reading and Using This Page	910
Access Control - defining entries	912
Privileges - create new or edit existing privileges	913
Artifacts	914
Artifact Details	915
General Information section	915
The "tabbed" sections	915
Artifact Versions table	915
Properties table	916
Artifact - create new or edit existing artifact	917
To create a new artifact	917
To edit an artifact	917
Artifact Versions	919
Artifact Version Details	920
General Information section	920
The "tabbed" sections	920
Files	920
Retrievers table	921
Dependent Artifact Versions table	921

Properties table	921
Artifact Version - edit an existing artifact version	922
To edit an artifact version	922
Repositories	923
Repository - create new or edit existing repository	924
To create a new repository	924
To edit a repository	924
Database Configuration	925
Defect Tracking Configurations	926
Defect Tracking - create new or edit existing configuration	927
To create a defect tracking configuration	927
Using the defect tracking integration	928
To edit an existing defect tracking configuration	928
Defect Tracking Reports	929
Email Notifier - create new or edit existing email notifier	930
To create a new email notifier	930
To edit an existing email notifier	933
Email notifier templates	933
Email Configurations	936
Email Configuration - create new or edit existing email configuration	937
To create a new email configuration	937
To edit an existing email configuration	938
Email Notifier - create new or edit existing email notifier	939
To create a new email notifier	939
To edit an existing email notifier	942
Email notifier templates	942
Email Notifier Template - Html_JobStepTempl_AllSteps.txt	945
Email Notifier Template - Html_JobStepTempl_SingleStep.txt	946
Email Notifier Template - Html_JobTempl.txt	947
Email Notifier Template - Html_StateTemplate_ApproveWorkflow.txt	948
Email Notifier Template - Html_StateTemplate_FullWorkflow.txt	949
Email Notifier Template - JobStepTempl_FullProps.txt	950
Email Notifier Template - JobTempl_FullProps.txt	951
Email Notifier Template - StateTemplate_FullPropertyPaths.txt	952
Event Log	953

Configuring the Event Log	953
Jobs	955
Jobs	957
Job Details	959
Summary section (at the top of the page)	959
The "tabbed" sections	961
Steps table	961
Diagnostics table	962
Parameters table	963
Properties table	963
Notifiers table	964
Published Artifact Versions table	964
Retrieved Artifact Versions table	965
Job Step Details	966
Summary section (at the top of the page)	966
The "tabbed" Sections	967
Child Steps table	968
General table	968
Diagnostics table	969
Parameters table	969
Properties table	969
Notifiers table	969
Job Step Time and WaitTime Properties Explained	970
Licenses	972
All Licenses	972
Current Usage section	972
Three License types	973
Concurrent Resource License	973
Concurrent User License	974
Concurrent Step License	974
View License	975
Import License	976
Plugin Manager	977
Using a plugin	979
Configuring the plugins directory	980
Procedure Details	981
Procedure Steps	981

Creating a new step	981
Parameters	982
Email Notifiers	982
Custom Procedure Properties	982
Procedure - create new or edit existing procedure	983
To create a new procedure	983
To edit an existing procedure	984
Step - create new or edit existing step	985
To create a new command or subprocedure step	985
To edit an existing command or subprocedure step	991
Publish Artifact Version Step	993
Retrieve Artifact Version Step	994
Artifact Retrieval Dependency Order Explained	995
Send Email Step	997
New Extract Preflight Sources Step	998
Parameter - create new or edit existing parameter	999
To create a new parameter	999
To edit an existing parameter	1001
Projects	1003
Projects have two purposes	1003
ElectricCommander includes default projects	1003
ElectricCommander solutions	1004
Project - create new or edit existing project	1006
To create a new project	1006
To edit an existing project	1007
Project Details	1008
The "tabbed" sections	1008
Procedures tab	1008
Workflow Definitions tab	1009
Jobs tab	1009
Workflows tab	1010
Schedules tab	1011
Credentials tab	1012
Properties tab	1012
Reports tab	1012
Run Procedure	1014

Using the Run Procedure page	1014
Schedule - create new or edit existing schedule	1016
To create a new standard schedule	1016
To create a new continuous integration "schedule"	1019
To edit an existing schedule	1019
Credentials - create new or modify existing credential	1020
To create a new credential	1020
To edit a credential	1020
Properties	1021
Overview	1021
Creating or modifying properties	1022
Using property values	1023
Property sheets and intrinsic properties	1023
Property names and paths	1024
Absolute property paths	1024
Relative property paths	1026
Context-relative "shortcuts" to property paths	1027
Property path shortcuts in ElectricCommander 5.0 and later	1030
Property name substitutions	1032
Expandable properties	1032
Custom property names and values	1032
The property hierarchy	1033
Special property references	1034
increment	1034
timestamp	1034
javascript	1035
Commander Intrinsic Properties listed by object type	1036
Commander Property Type definitions	1037
Commander Object / Property tables	1038
Property error codes	1100
Property - create new or edit existing property	1103
Nested Property Sheet	1104
Reports	1105
Create a new report	1106
Multiple Series Report	1106
Category Report	1107
Count Over Time Report	1108

Procedure Usage Report	1109
Resources	1111
Commander supports two types of resources	1111
Resource page information and functions	1112
New Resource panel	1118
New Proxy Resource panel	1122
Edit Resource panel	1126
Edit Proxy Resource panel	1126
Resource Details panel	1126
Switching a non-trusted agent to trusted	1127
Install or Upgrade Remote Agents	1127
Prerequisites:	1128
Using the Install or Upgrade Remote Agents dialog	1128
Resource Pools	1137
Resource Pool - create new or edit existing pool	1138
To create a new resource pool	1138
To edit an existing resource pool	1140
Unusable resources	1140
Resource Pool Details	1141
The "tabbed" sections	1141
Properties tab	1142
Zones	1144
Zone Details panel	1145
Creating a new zone	1145
Access Control notes	1145
Gateways	1147
Gateway Details panel	1148
Create Gateway panel	1148
Edit Gateway panel	1149
Access Control note	1149
Define Search	1151
Search Results	1152
Server	1153
Settings - edit existing property settings	1154
Source Control Configurations	1157
Source Control Configurations - create new or edit existing configuration	1158

AccuRev	1158
ClearCase	1159
File	1159
Git	1159
Perforce	1159
Property	1160
Subversion	1160
Checking out code from a source control system	1161
To edit an existing source control configuration	1161
The Home Page	1162
Overview	1162
Job Configurations	1162
Create Job Configurations three ways	1162
Shortcuts	1163
Create Shortcuts two ways	1163
Jobs Quick View	1163
Create a job category	1163
Reports	1164
The Home Page	1165
Overview	1165
Job Configurations	1165
Create Job Configurations three ways	1165
Shortcuts	1166
Create Shortcuts two ways	1166
Jobs Quick View	1166
Create a job category	1166
Reports	1167
Job Configuration	1168
To create a new job configuration	1168
To edit an existing job configuration	1169
Shortcuts	1170
Jobs Quick View	1171
To create a new jobs quick view category	1171
To edit an existing jobs quick view category	1172
New Report	1173
To populate the drop-down menu... ..	1173
Active Users	1174

Users and Groups	1175
User - create new or edit existing local user	1177
To create a new local user	1177
To edit an existing local user	1177
User Details	1178
Edit User Settings	1179
Groups	1180
Group - create new or edit existing local group	1181
To create a new local group	1181
To edit a local group	1181
Group Details	1182
Directory Providers	1183
Directory Providers - create new or edit existing directory providers	1184
Commander supports Active Directory and LDAP directory providers	1184
To create a new Active Directory provider	1184
To create a new LDAP directory provider	1187
Examples for directory provider field descriptions	1190
To edit an existing directory provider	1191
Test Directory Provider	1193
Workflows	1194
Workflow Definition - create new or edit existing workflow definition	1195
To create a new workflow definition	1195
To edit an existing workflow definition	1196
Workflow Definition Details	1197
Graph view	1198
To create a new state	1199
To create a new State Definition	1200
To create a Transition Definition	1204
Using the graph's "quick-access" features	1206
Show Legend	1207
List view	1208
Properties view	1211
Run Workflow	1212
Workflow Details	1213
Summary section - at the top of a running workflow page	1214
When the workflow is complete...	1215

Graph tab	1216
Show Legend tab	1217
Using the graph's "quick-access" features	1217
Show History	1218
States tab	1220
State Details panel	1220
Parameters tab	1223
Properties tab	1223
Workflow Log	1225
Transition Workflow	1227
Workspaces	1228
Workspace - create new or modify existing workspace	1230
To create a new workspace	1230
To edit an existing workspace	1231
Workspace file	1232
Tutorials	1233
Adding a link to a job	1234
Implementation	1234
Related information	1235
Calling a subprocedure	1236
Implementation	1236
Related information	1237
Checking outcome of preceding step	1238
Implementation	1238
Related information	1239
Conditional execution	1240
Implementation	1240
Related information	1241
Custom parameter layouts	1242
Implementation	1243
Related information	1243
Email notifications	1244
Implementation	1244
Related information	1246
Executing tasks on all resources in a pool	1247
Implementation	1247

Related information	1247
Factory procedures	1249
Implementation	1250
Related information	1251
Postp extension	1252
Implementation	1253
Related information	1253
Publishing and retrieving an artifact	1254
Implementation	1255
Related information	1255
Reserving a resource for job step duration	1256
Implementation	1256
Related information	1256
Running steps in parallel	1258
Implementation	1258
Related information	1259
Step timeouts and steps that always run	1260
Implementation	1260
Related information	1260
Storing and retrieving properties in a job	1261
Implementation	1261
Related information	1261
Working with properties stored in a procedure	1263
Implementation	1264
Related information	1264

Welcome to ElectricCommander

[What makes Commander unique?](#)

[Commander architecture](#)

Overview

ElectricCommander is an enterprise-class solution for automating the software build, test, and release process. Commander helps teams make software build, package, test, and deployment tasks more repeatable, more visible, and more efficient.

At its core, ElectricCommander is a web-based system for automating and managing the build, test, 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
 - little or no management visibility or reporting
- Multiple, disconnected build and test systems across locations, resulting in:
 - redundant work
 - inability to share/reuse code files across teams
 - painful to manage build and test data
- Slow overall build and release cycles that directly impact:
 - release predictability
 - time-to-market

Commander tackles these problems with 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/reporting for better project predictability
- Better software quality by integrating and validating against all target platforms and configurations

What makes ElectricCommander unique?

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

Commander 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.

Commander's unique analytics provide visibility into one of the best indicators of project success: compiled, tested, working code. Commander'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 as well as build trends by project and resource utilization. Commander's integration with virtual lab automation (VLA) solutions also allows you to snapshot or reproduce a specific build for auditing or troubleshooting purposes.

Commander provides unified process automation across the entire build/test/deploy lifecycle 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, Commander 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.

What you do

To use Commander, you need to:

- Configure *resources*: these are machines Commander can use to run jobs.
- Define *procedures*: these are the tasks you want to automate and contain one or more steps.
- Create *schedules*: these define when each procedure will run.

What Commander does

After setting up resources, procedures, and schedules, Commander takes over to run the procedure you created. Here are some of the facilities ElectricCommander provides:

- Zones and Gateways - A zone (or top-level network) 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 to another zone. For example, you might want a developers zone and a test zone. The Commander server is a member of the *default* zone, created during Commander installation.

- *Continuous Integration Builds* and other schedules: Run jobs according to *schedules* you define. Scheduled jobs can run at specific times or when source code changes are checked in to your source control system. Commander 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 more.
- Provides *Artifact Management* functionality: Using artifacts can improve performance across builds, provide better reusability of components, improve cross-team collaboration with greater tractability. For example, instead of developer repeatedly downloading third-party packages from external sources, these components can be published and versioned as an artifact. A developer then simply retrieves a specific artifact version from a local repository, guaranteeing a consistent package from build to build.
- Provides *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: Commander is built with an extensible UI, enabling easy development of plugins, including integrations with other tools, custom dashboards, and unique user experiences based on roles. "Bundled" plugins, installed during Commander installation provide easy integration with your SCM systems, defect tracking applications, and much more.
- Provides *Workflow* functionality: Use a Workflow to design and manage processes at a higher level than individual jobs. Workflows allow you to combine procedures into processes to create build-test-deploy lifecycles (for example). A workflow contains states and transitions you define to provide complete control over your workflow process. The Commander Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs.
- Handles *resource* management: If a resource is overcommitted, Commander delays some jobs until others are finished with the resource. You can define pools of equivalent resources and ElectricCommander spreads usage across the pool.
- Records a variety of information about each job, such as the running time and success or failure of each step. A set of reports is available to provide even more information.
- Provides powerful and flexible *reporting* facilities. Various statistics such as number of compiles or test errors are collected after each step and recorded in the ElectricCommander database. A variety of reports can be generated from this information.
- Allows you to observe jobs as they run and to cancel jobs or change their priorities.
- Provides a *workspace* for each job, which is a disk area a job uses for storage. Commander also provides a facility for reclaiming space occupied by workspaces.
- Provides a powerful data model based on *properties*. Properties are used to store job input data such as which source code branch to use for the build, and 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).
- Provides *access control* for users logged into the system and Commander uses this information to control their activities. Commander integrates with Active Directory and LDAP repositories.
- Provides *search, sort, and filter* functions to minimize viewing or "wading" through information that is of no interest to you, allowing you to get access to the information you need quickly.
- Provides *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 Commander operations and features are available from a command-line application tool, *ectool*, as well as a web interface.

Schedules

Schedules are used to execute procedures and determine when specific procedures run. A schedule can trigger at defined times, for example, every 2 hours from 10 pm to 6 am on Mondays, Wednesdays, and Fridays, or when modifications are checked into a particular branch of your source code control system. Also, it is 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 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 lifecycle spanning multiple procedures and projects requires a significant amount of "meta-programming" and a heavy use of properties – the *workflow* feature simplifies this process. Using the Commander workflow object, you can create "build-test-deploy" lifecycles by defining a set of states and transitions. Any Commander project can contain a workflow.

Projects

A *project* is an object used in Commander to organize information. A project is a container object for procedures, steps, schedules, workflows, and properties. If you use Commander for different purposes, you can use a separate project for each purpose so different uses 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.

Procedures and Steps

Procedures and *steps* define tasks you want Commander 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 Commander 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 Commander 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, Commander 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. (Another name for a nested procedure is "subprocedure".)

Jobs

A *job* is a Commander object that is created each time a procedure begins to execute (or "runs"). 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. Commander retains job information after the job completes so you can examine what occurred.

Resources

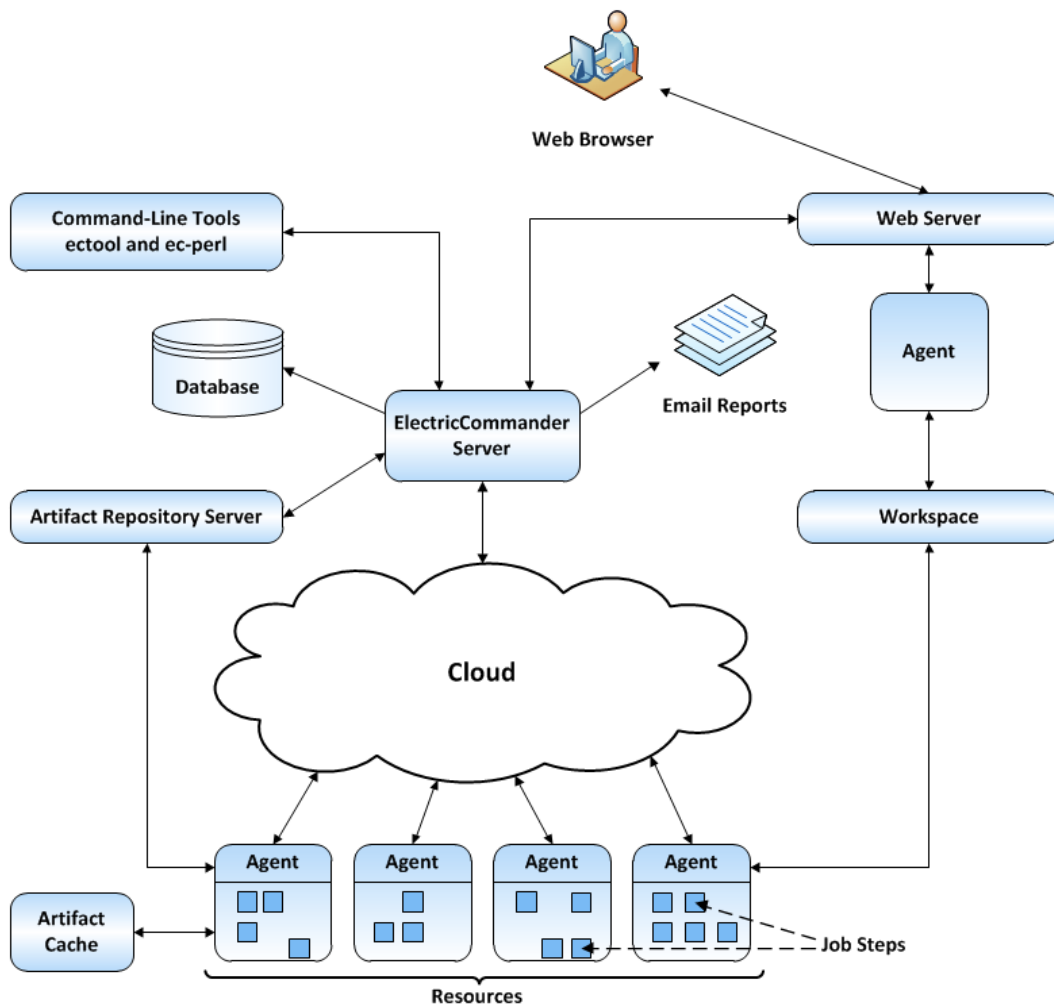
A *resource* is a server machine available to Commander for running steps. A resource has a logical name in addition to a host name. In some situations, it is convenient to have multiple logical resources associated with the same host. Also, a resource can be associated with one or more pools. Each resource 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*.

Commander architecture

ElectricCommander was architected from the ground up to support small, mid-range, or enterprise scale software production. Based on a 3-tier architecture, Commander scales to handle complex environments. The Commander multi-threaded Java server provides efficient synchronization even under high job volume.

- The Commander server manages resources, issues commands, 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.

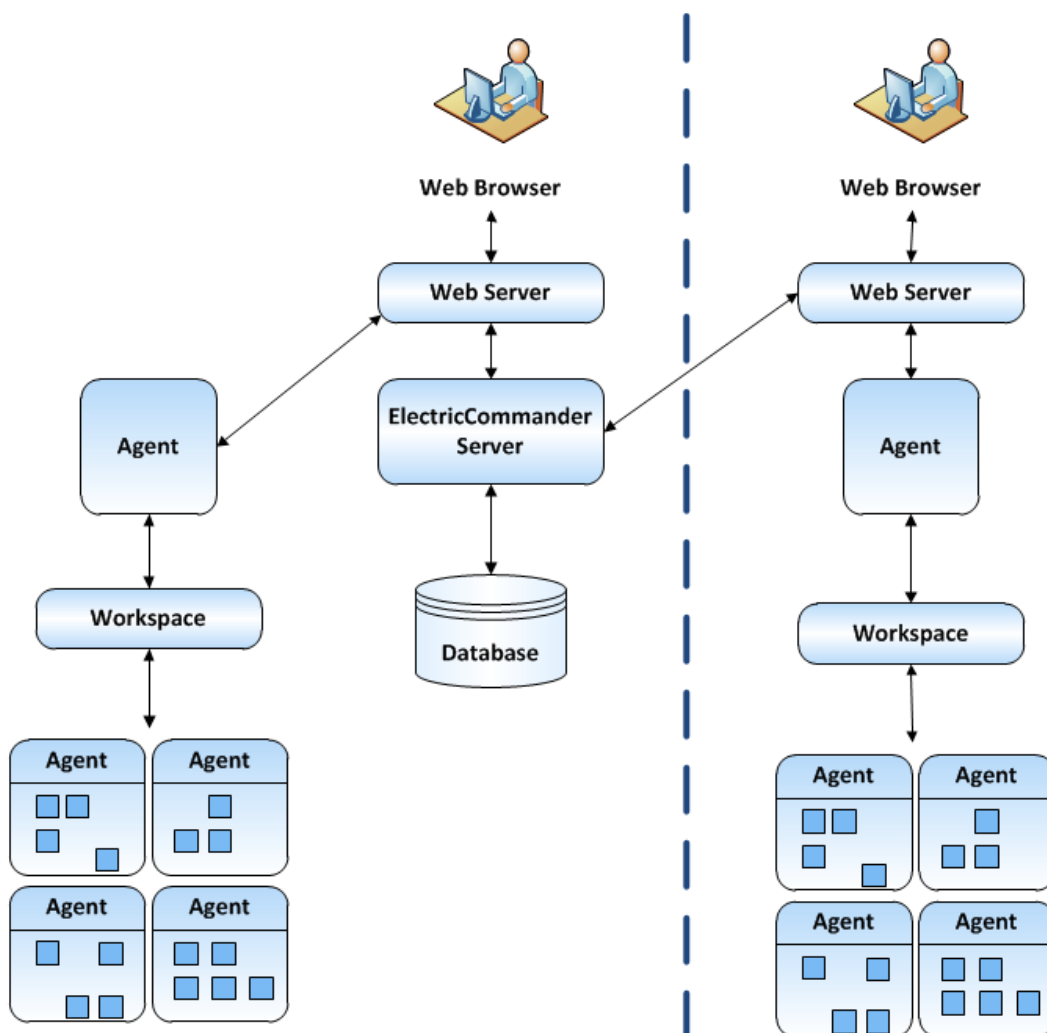
Simple architectural overview



Expanded remote configurations

ElectricCommander is not limited by only the components illustrated above. The following screen example illustrates a remote web server configuration.

The next illustration is an example for how you might set up a remote web server installation.



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

Other possible configurations

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

Getting Started

[Basic Commander terminology](#)

[Getting Started Scenarios help topic series](#)

[Navigating the Commander user interface](#)

Overview

ElectricCommander is an application for automating and managing the software build, test, and deployment process—your development lifecycle. Commander helps development teams make these tasks repeatable, visible, and efficient.

Commander creates an environment where IT and Development organizations can work together to connect physical and virtual environments, processes, and tools already in use to create a private development cloud—we call this a “smart development cloud”. Commander is a scalable solution, solving some of the biggest challenges of managing “back-end” software development tasks.

Basic Commander terminology

While working through the scenarios, you may encounter unfamiliar terminology. Understanding Commander terminology will acquaint you with Commander concepts and processes, providing an overview of how Commander works for you.

Object	Description
Project	Commander 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.
Procedure	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.
Step	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.

Object	Description
Plugins	Commander is built with an extensible architecture, enabling easy development of plugins, including integrations with other tools, custom dashboards, and unique user experiences based on roles. Numerous plugins are installed during Commander installation, which makes them transparent to the user. For example, in Scenarios 1 and Scenario 2 help topics, you will configure a source control system and build an Ant step, both of which use plugin technology.
Parameter	This is a value you pass to a procedure that controls procedure operation.
Schedule	A standard schedule defines when your procedure will run. You can control the hour/day your procedure runs or use a trigger to run a procedure each time a particular event occurs.
Continuous integration	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 may want to run a procedure to build your software every time you check in code.
Resource	A resource is an agent machine configured to communicate with Commander. Your steps can run on any resource or resource pool you define. Resources can be grouped into a “pool” to increase your step’s processing speed by assigning steps to less loaded members of the pool. Also, pools can be used to ensure the resources you need are available to a specific group of developers only.
Job	These record the results of running a procedure. This means each time you run a procedure, a job is created—you can view the results of running a procedure and each step within the procedure. Tables are provided on the Job Details page to view or manage your results, or for future reference.
Report	Reports display your information graphically for review or to show trends and more.

Object	Description
Workspace	Commander provides a default area on the disk 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 within its workspace, such as step logs, and Commander 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 within a job to use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.
Zones	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—agents in one zone cannot directly communicate with agents in another zone.
Gateways	To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the <i>source</i> zone and the other in the <i>target</i> zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Getting Started Scenarios help topic series

The series of scenario help topics are designed to help you learn to use Commander quickly. The step-by-step scenario format “walks” you through learning Commander basics:

- creating a simple procedure - [Scenario 1](#)
- creating a procedure that uses a Source Code Management (SCM) system - [Scenario 2](#)
- setting up email notifications, reporting, and scheduling - [Scenario 3](#)
- creating a multi-agent build procedure - [Scenario 4](#)

Each scenario builds on what you learned in the previous scenario. And each scenario ends with one or more scenario “extensions” to introduce you to other related Commander functionality.

Navigating the Commander user interface

To quickly link to task pages, view running or completed jobs, configure resources and more, the Commander web interface displays tabs across the top of the page—these tabs remain available at the top of all Commander web pages.

When you select some of the main tabs, subtabs are provided. For example (see the next screen example), when the **Home** tab is selected, notice that the Overview tab is in bold font—this is because you are looking at the Overview page. This page can be customized to show the information you want to see quickly. Use this page for shortcuts, quick links to jobs, "thumbnail" report views, and more.

Selecting the Continuous Integration subtab takes you to the Continuous Integration Dashboard, which is the place to configure your projects for continuous integration builds.

In addition to the main set of tabs, the top bar includes:

- **Logged in as** clearly shows the name of the logged-in user.
- **Logout** - Use this link to logout between Commander sessions if necessary.
- **Help** - This link provides a help topic for the current page you are viewing, but if you select the down-arrow adjacent to the Help link, you have access to:
 - **Tutorials** - use this link to display the Table of Contents for all currently available tutorial examples you may find useful as you become familiar with Commander.
 - **Welcome Screen** - this link displays the Welcome Screen again in the event you would like to revisit available Commander videos or other product information.
 - **Documentation** - use this link to access the entire Commander online help system, which is fully searchable in the event you do not quickly see what you are looking for in the left-pane Table of Contents.

The Help Table of Contents contains feature overview/concept user-guide style help topics, which are not linked to a particular web page. These topics are listed in the Table of Contents *above* the "Web Interface Help" section. The Web Interface Help section contains all of the page-specific help topics.

Note: *To print a Help topic:* Either right-click your mouse in the Help topic pane or go to your browser's File menu.

- **Visit the Community Site** - this link takes you to the Electric Cloud Knowledge Base, all product documentation, and more.
- **Contact Support** - use this link to access the Electric Cloud Technical support phone number and email address.
At a later time, you may choose to re-configure this link to re-direct to your internal support team. If so, see the "Customizing the Commander UI" Help topic, [Reconfigure Contact Support link](#) section.
- **About** - use this link to display the current version of Commander you are using.

The Home tab

You can customize the Home page to see the information you need at-a-glance or add Shortcuts to quickly return to a Commander pages you visit frequently. The Home page provides 4 categories:

- **Job Configurations**
Commander procedures can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem.

- **Shortcuts**

Use shortcuts to save frequently visited Commander web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

- **Jobs Quick View**

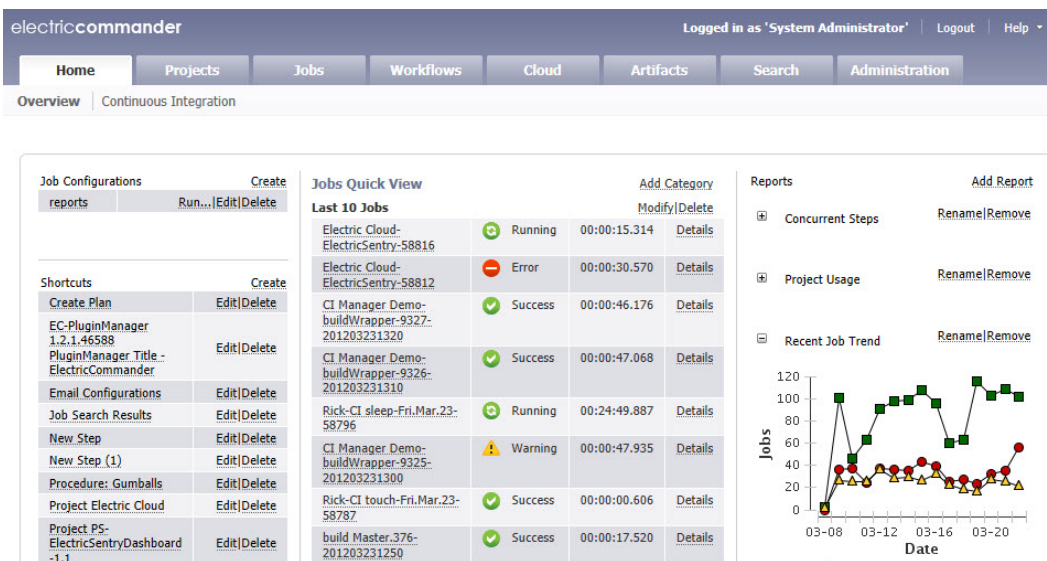
The Jobs Quick View allows you to define job categories that are interesting to you. Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

- **Reports**

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

On the Home page example below, notice other additional links. This page allows you to create, edit, and delete objects you see so you can easily keep the page updated and current.

For more information about Home page functionality, see [The Home Page](#) help topic.



The Continuous Integration subtab

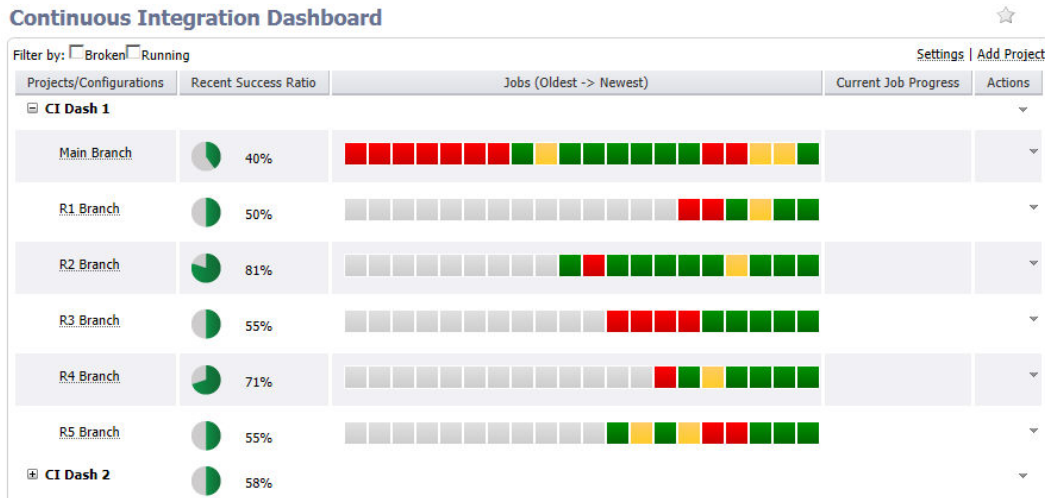
The Continuous Integration Manager (CI Manager) provides a front-end user interface (the Continuous Integration Dashboard) for creating, managing, and monitoring continuous integration builds.

The Continuous Integration Dashboard provides:

- Visually see your running builds, build progress, build status, and accumulated build status.
- Easily accessed "Actions" to configure a continuous integration build.
- Quick configuration of your preferred SCM system.

- A project can contain any number of continuous integration builds, depending on the work you have already setup for your procedures/steps to perform.

The following is an example of the dashboard, but your initial view will be "blank" until you add configurations (using your preferred SCM) to run your build procedures continuously. When this tab is opened to view the dashboard, click the **Help** link in the upper-right corner of the page for instructions to begin using the dashboard.



Projects, Jobs, and Workflow tabs

Selecting the Projects, Jobs, or Workflows tab displays a table listing all previously configured projects, all completed jobs, and all previously configured workflows, respectively. Each table contains links within the table as well as links in the section above the table.

For example, selecting the Projects tab displays table (see the next screen example), which contain projects you created and Commander default projects, added during Commander installation.

- Links at the top of the table include:
 - A "star" icon - click this icon to add this page to your Home page.
 - A drop-down menu to choose which projects you want to see.
 - Create Project link to go to the New Project page to define and add a new project.
 - New Search link to find a project whose name you may not quite remember or projects of the same type.
- Links within the table include:
 - Click on a Project name (first column) to go to the Projects Details page for that project.
 - Actions column - this column generally contains links to Edit, Copy, or Delete the object (a "project" in this example).

Again, each table—Projects, Jobs, or Workflows—contains similar links to help you get more information, modify existing information, or create a new object. For more information on any of these Commander pages, see their respective Help topics: [Projects](#), [Jobs](#), or [Workflows](#)

Projects

63 Results All Projects Create Project New Search

Project	Description	Impersonation Credential	Create Date	Actions
CVS			2012-02-10 09:06:29 PST	Edit Copy Delete
Default	Default project created during installation.		2011-09-13 16:42:30 PDT	Edit Copy Delete
Default copy	Default project created during installation.		2011-09-13 16:42:30 PDT	Edit Copy Delete
DependencyManager-1.1	Manage dependencies between targets or components. See Dependencies Documentation for more information.		2011-12-27 15:59:11 PST	Edit Copy Delete
Dimensions Project			2012-02-14 09:35:44 PST	Edit Copy Delete
EC-Examples	This project contains templates for procedures that perform basic tasks within ElectricCommander.		2011-09-13 16:42:33 PDT	Edit Copy Delete

Cloud tab

This tab opens to the Resources page and provides several other subtabs—Pools, Workspaces, Zones, Gateways, and the Cloud Manager plugin if it is installed.

The Resources page

(example below) This page displays all resources you configured for Commander to use.

- Notice the information immediately after the "Resources" title at the top of the table - you can see how many licensed resources are in use.
- The row of icons at the top of the table allow you to perform numerous resource management functions.
- A resource name, in the Name column in the table, is a link to open that resource's Resource Details panel for more information about the resource, and to access the Edit Resource panel to make modifications or add access control privileges.

See the [Resources](#) Help topic for more information.

electriccommander Logged in as 'System Administrator' | Logout | Help

[Home](#) [Projects](#) [Jobs](#) [Workflows](#) **Cloud** [Artifacts](#) [Search](#) [Administration](#)

[Resources](#) [Pools](#) [Workspaces](#) [Zones](#) [Gateways](#) [Cloud Manager](#)

Resources — Current Usage: 0 of 100 Managed Hosts; 0 of 100 Proxied Hosts

[New Search](#)

Filters [Save Filters](#) [Reset](#)

Status

Both Enabled/Disabled

Step Limit

Proxy Agent

[Filter](#)

		Name	Pools	Job Status	Description	Zone	HTTPS & Host	Load
<input type="checkbox"/>		A1	P1		test resource	default	localhost	
<input type="checkbox"/>		A2				default	localhost	
<input type="checkbox"/>		A4			This agent has not been pinged yet	default		
<input type="checkbox"/>		AD1			This agent has not been pinged yet	default		

The Pools subtab

Click this subtab to display a list of all resource pools available to Commander. You may find it useful to group resources, creating one or more resource pools, perhaps for specific purposes, Development teams, or other groups in your organization.

Similar to the Resources page, this page provides:

- A link to create a new resource pool.
- A Search link to find an existing resource pool.
- Summary information for each resource pool in the list.
- Clicking the resource Pool Name takes you to the Resource Pool Details page for more information about that pool.
- And the Action column contains Copy and Delete links.

For more information about pools, see the [Resource Pools](#) or [Resource Pool - create new or edit existing pool](#) help topics.

The Workspaces subtab

Similar to the Resources and Pools pages, the Workspaces page displays a table all available workspaces and summary information for each workspace.

Available links are:

- **Create Workspace** for adding another workspace.
- **New Search** to find an existing workspace.
- Clicking the Workspace Name takes you to the Edit Workspace page to make modifications.
- And the Action column contains **Copy** and **Delete** links.

For more information about workspaces, see the [Workspaces and Disk Space Management](#), [Workspaces](#), or [Workspaces - create new or edit existing workspace](#) Help topics.

The Zones and Gateways subtabs

Similar to the other subtabs under the Cloud tab, the Zones and Gateways subtabs each display a table, listing all available zone or gateways, respectively.

- Selecting an object in the Name column takes you to that object's "Detail" page for more information.
- These pages are your management centers for zones or gateways, respectively.

For more information, see the [Zones](#) and [Gateways](#) Help topics.

Artifacts and Search tabs

This page displays all artifacts available on this Commander server and like the Projects, Jobs, and Workflows pages, Create, Edit, and Delete links are provided for the object—in this case, an artifact. Clicking on an artifact Name (first column) takes you to the Artifact Details page. *For more information*, see the [Artifact Management](#) Help topic.

Name	Group Id	Artifact Key	Description	Actions
bar:bar	bar	bar		Edit Delete
bar:foo	bar	foo		Edit Delete
EC-Tutorials:MyArtifact	EC-Tutorials	MyArtifact		Edit Delete
foo:bar	foo	bar		Edit Delete
My:Artifact	My	Artifact		Edit Delete

Records per page: 20 1 thru 5 of 5

The Artifact Versions subtab

The Artifact Versions page displays all artifact versions available on this Commander server. Links within the table are the similar to other Commander pages of this type. For more information on this page, see the [Artifact Versions](#) Help topic.

The Repositories subtab

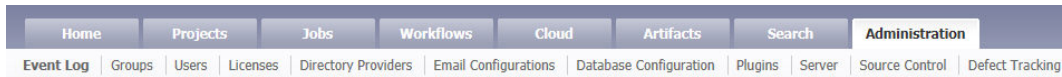
The Repositories page displays all artifact repositories available to this Commander server. For more information about repositories, see the [Artifact Management](#) and [Repositories](#) Help topics.

Search tab

While many Commander web pages offer a Search link, some do not, but this tab is always available. This page allows you to search for any Commander supported object type. For more information, see the [Define Search](#) Help topic.

Administration tab

The next screen example illustrates all the current subtabs available when you select the Administration tab. Less-frequently used tasks are grouped for you [as subtabs] under the Administration tab. For example, after users and groups are set up, you do not need these pages unless a user or group configuration changes because you hired a new employee, someone's job responsibilities changed, or someone left the company.



Administration subtab descriptions:

- **Event Log** - This page displays a log of events generated anywhere in the system, including jobs and workflows.
- **Groups** - Use this page to view a filtered list of Commander *local* users or groups.
- **Users** - Use this page to view a filtered list of Commander *local* users or groups.
- **Licenses** - This page displays all license information known to the Commander server. Typically, a single license is displayed, which describes the usage to which you are entitled. The server may be licensed by concurrent resources, concurrent users, or both.
- **Directory Providers** - Commander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. **Local** users and groups are defined within Commander.
- **Email Configurations** - This page displays all previously configured email configurations. You must create an email configuration so Commander can communicate with your mail server if you intend to send email notifications
- **Database Configuration** - If you chose not to use the Commander-supplied default database, use this page to configure another database to communicate with Commander. See the *ElectricCommander Installation Guide* for a list of approved databases for use with Commander.
- **Plugins** - This subtab displays the Plugin Manager page. A *plugin* is a collection of one or more features that can be added to Commander. Numerous plugins are bundled and installed with Commander, integrated third-party products seamlessly with Commander. For example, Source Code Management plugins are installed and waiting for you to configure your preferred SCM to communicate with Commander.
- **Server** - This page displays overall information about the Commander server. Use the Settings or Access Control links to make a few modifications you might need.
- **Source Control** - This page displays all SCM (source code management) configurations you have created to communicate with Commander.
- **Defect Tracking** - This page displays previously configured Defect Tracking systems known to the Commander server—Commander integrates, using plugins, with numerous defect tracking systems.

Go to [Scenario 1](#)

Getting Started - Scenario 1 - Creating a simple procedure

To begin, select the Projects tab, then the **Create Project** link at the top of the table. You need to create a project to contain the procedures you create.

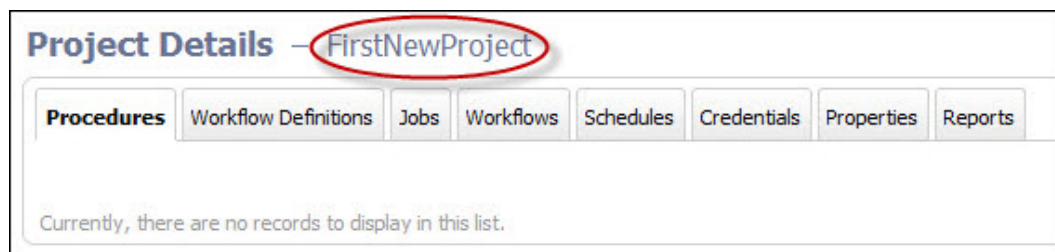
On the New Project page, fill-in the fields as follows:

Field Name	Description
Name	Type a unique project name. You may want your project names to reflect the work groups or teams that will be using them. For example, you might set project names based on the products they support. For our scenario examples, we will use "FirstNewProject" for our project name.
Description	Supply a text description for your reference if you choose. ElectricCommander does not use or interpret this information.
Default Resource	(optional) For the purpose of "getting started", leave this field blank to use the Default resource already created during the Commander installation.
Default Workspace	(optional) For the purpose of "getting started", leave this field blank to use the Default workspace already created during the Commander installation.

Click **OK** to save the information you entered.

After clicking **OK**, you see the Project Details page and the name of the new project adjacent to the page title.

- For our scenarios, the project name is FirstNewProject.
- Your new project name will appear on the Projects page also.



Scenario requirement: No additional requirements for this scenario. This scenario uses the *local* agent (resource) that was included on your local machine when Commander was installed.

Overview

This scenario establishes that Commander and its components were installed successfully and guides you through creating a simple procedure with a parameter that will echo Hello World. After you create this procedure, you will run the procedure and review the results.

At the end of this scenario, you will be familiar with the following Commander concepts and features:

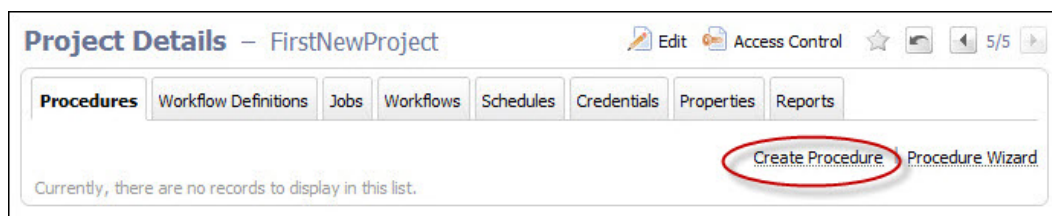
- Projects
- Procedures
- Steps
- Parameters
- Jobs
- Navigating the Job Details web page

Begin Scenario 1

Step1. Select the Projects tab

On the Projects page, you can see the default projects installed during the Commander installation and the new project you just created.

Select your new project name to go to its Project Details page, then click the **Create Procedure** link.



Step 2. Create a procedure

On the New Procedure page, you need to be concerned with the section of the page illustrated below only.

Supply the following information:

Field Name	Description
Name	For this scenario, <code>Hello World</code> is the procedure name. At a later time when you are creating your own procedures, you can choose any unique name that is most meaningful for procedures in your project.
Description	(optional) Supply a text description for your procedure if you choose to do so.
Default Resource	Use the Browse link and choose <code>local</code> . Using <code>local</code> accesses the agent installed during the Commander installation. At a later time after you have configured other resources, your Browse list could contain many choices.

Your New Procedure page will look similar to the following:

Project: [FirstNewProject](#)

New Procedure

Name:

Description:

Job Name Template:

Default Resource: [Browse](#)

Before leaving this page, notice the "breadcrumb" in the top-left corner. Most Commander web pages contain a breadcrumb for easy return to the previous page.

Click **OK** to continue.

After clicking **OK**, Commander takes you to the Procedure Details page—this is the page you use to add important information to your procedure. When your procedure contains steps, parameters, or email notifiers, and so on, those objects will be displayed here for your reference.

Step 3. Create a parameter

A Commander procedure can define one or more parameters. These parameters are similar to the parameters of a subroutine in a programming language. Parameters allow you to define a "re-usable" procedure by using

symbolic variable names in place of a single, fixed value. When you run the procedure, you can type in a value to use for that particular job. You can refer to the parameter in a step for a procedure, using the standard Commander "property reference syntax" - `$(...)`.

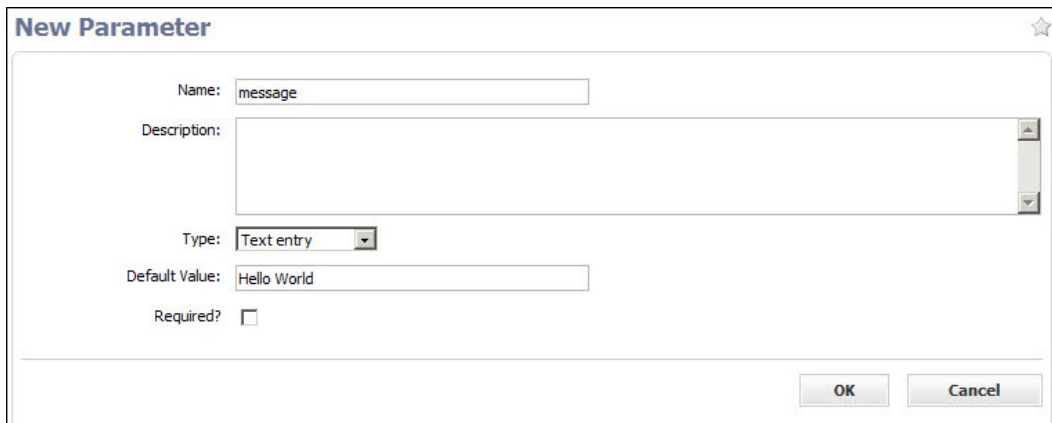
For example, see **Step 4**, specifically the text in the Command(s) text box.

On the Procedure Details page, click the **Create Parameter** link to go to the New Parameter page.

Supply the following information:

Field Name	Description
Name	Supply the parameter name <code>message</code> for this scenario.
Type	Use the default <code>Text entry</code> for the parameter type.
Default Value	Type-in <code>Hello World</code> for the default parameter value.

Your New Parameter entries will look like the following:



Click **OK** after filling in the fields and Commander returns you to the Procedure Details page.

On the Procedure Details page:

Notice the Parameters table is populated with your parameter entry.

Procedure Details - Hello World

Run Edit Access Control

Procedure Steps New Step: [Command](#) | [Subprocedure](#) | [Plugin](#)

Currently, there are no records to display in this list.

Parameters [Create Parameter](#)

Parameter Name	Default Value	Type	Req?	Description	Actions
message	Hello World	entry			Delete

Email Notifiers [Create On Start Email Notifier](#) | [Create On Completion Email Notifier](#)

Currently, there are no records to display in this list.

Custom Procedure Properties [Create Property](#) | [Create Nested Sheet](#) | [Access Control](#)

Property Name	Value	Description	Actions
ec_customEditorData			Edit Delete

Also notice the name of the procedure we created (Hello World) is shown adjacent to the page title for your reference.

Step 4. Create a step

Now we can create a step on this procedure. On the Procedure Details page, to create a New Step for our current purpose, click the **Command** link to go to the New Step page.

On the New Step page:

The following screen displays the portion of the New Step page you need for this scenario.

Notice this version of the New Step page contains a Command text box to fill-in the information you need. At a later time, when you choose to create a different step type, the New Step page will be populated with the appropriate fields required for that step type.

Supply the following information:

Field Name	Description
Step Name	Use <code>Print message</code> for the step name. For work outside this scenario, you can supply any unique name of your choice for the step name.
Description	(optional) Add a text description about this step if you need one.

Field Name	Description
Command(s) box	<p>Type <code>echo \${message}</code></p> <p>In this command, <code>message</code> refers to the parameter we created.</p> <p>Note: The Commander Server does not actually interpret (or even understand) the contents of this field. Commander simply expands all property references and then passes the resulting text block to the Commander Agent. The Agent copies the text block to a temporary file and then invokes the Agent's native shell command, passing the name of the temporary file as a parameter to the shell command. The overall result is the same as if you had created this block of text as a BAT file (on Windows) or a shell script (on Linux). The operating system of the Agent machine takes over from that point to actually run your commands.</p>
Resource(s)	<p>For this scenario, leave this field blank to use the default resource you set on the procedure. If multiple resources are configured, you have the option to change the resource for each step if you prefer.</p>

Your New Step page will look like the following example:

New Step

General

Name:

Description:

Command

Command(s):

Resource: [Browse](#)

Postprocessor:

Click **OK** to create the step and Commander returns you to the Procedure Details page to see its entry as displayed in the next screen example.

Procedure Details – Hello World Run Edit Access Control

Procedure Steps New Step: Command Subprocedure Plugin

Step Name	Resource	Action	Parallel	Time Limit	Actions
Print message		echo \${message}			Copy Delete

Parameters Create Parameter

Parameter Name	Default Value	Type	Req?	Description	Actions
message	Hello World	entry			Delete

Step 5. Run the procedure

Notice the green-circle-arrow Run icon among the links at the top of the tables. If you click the **Run** link, the procedure will run as-is immediately.

For this scenario, hover your mouse over the adjacent down-arrow and select the **Run...** option.

Selecting this option provides the Run Procedure page where you can add or modify the parameter value(s) as necessary.

Run Procedure – Hello World

Parameters

message: Default: "Hello World"

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

The Run Procedure page has the information we want for this scenario, so click the **Run** button.

When a procedure runs, it creates a "job".

After clicking **Run**, Commander takes you to the Job Details page to see the status of the "job" and watch its running progress.

This web page supplies a wealth of other information.

Job Details – job_14_201103251135

Run Again ▾ Delete Save Configuration ☆ >>

Completed with Success

Start Time: 2011-03-25 11:35:19 PDT
Elapsed Time: 00:00:02.197

General Information

Project: FirstNewProject
Procedure: Hello World
Launched by: admin
Priority: normal

Steps Diagnostics Parameters Properties Notifiers

View: All ▾ Expand All Collapse All

Step Name	Log	Status	Elapsed Time	Resource	Actions
Print message		Success	00:00:01.144	local	Edit

The Job Details page contains two sections. The section at the top of the page is the job's summary, displaying the job's status and other general information:

- The "green check" icon is displayed to show the job completed successfully.
- The "Completed with Success" section displays the result of the procedure that ran.
- The General Information section displays the project and procedure names and who launched the job.

The section below the summary area displays detailed job information, including tabs to expand information and links to access various other Commander web pages.

- Notice the "Success" entry in the Status column—this is your step result.
- In the Log column, click the icon in the Print message row to see the outcome/result of your Hello World Print message step. Your file will look similar to the following:

```
Job: job_14_201103251135

Workspace File – Print message.93.log

Hello World
```

- Select the Parameters tab to see the parameter you created and its value.

For complete information about the Job Details page functions and available information, click the **Help** link in the top-right corner of this Commander web page.

Scenario extension - Adding another step

Each procedure you create can contain as many steps as you need. We created only one step in this scenario. This example shows how to modify a procedure by adding additional steps. You can always add more steps or edit an existing step before you run a procedure again.

To add a step to an existing procedure:

1. On the Job Details page, select your procedure name (at the top of the page) to go to the Procedure Details page.
2. On the Procedure Details page, click the **Command New Step** link again.
3. On the New Step page, fill-in the fields as follows:

- **Name** - Print Step Info
- **Command(s)** - supply the following text:

```
print "This step was launched by ${launchedByUser} on the resource
${/myResource/resourceName}.\n";
```
- **Shell** - ec-perl
- Click **OK** to save the step and return to the Procedure Details page.

Note: You can move this new step to the top of the procedure by hovering your mouse over the icon next to the step name, then drag-and-drop the step to the top of the list. This is a useful function particularly if you find you have created steps in a different order in which you need them.

Click **Run** (from the Procedure Details page).

Finding a Commander web page...

In case you "lose" the Job Details page or any other Commander web page, several navigation methods are available:

- Use your browser's Back button if you were recently on a page you want to revisit.
- Use Commander "breadcrumbs." On some pages you see breadcrumbs on the left-side of the page that note your current position/level within the project.
- Use the tabs at the top of the page to drill-down to the page you need.

For example:

- **To find the Job Details page:**
Select the Jobs tab and notice that your procedure name is now listed in the Job column, accompanied by a date, job number, and other related job information on the same row. When you run a procedure, it becomes a job. Select the job you want to see and you will go to the Job Details page for that job.
- **To find the Procedure Details page:**
Select the Projects tab, then select your project to go to the Project Details page to see a list of procedures in that project. Click on a procedure to go to the Procedure Details page for that procedure.

Note: A "project" is a container for all related procedures you create. You can create as many projects as you need, using the **Create Projects** link on the Projects page.

As you use Commander, you will quickly learn other navigation paths for your particular interests. In addition, most Commander web pages contain a "star" icon in the upper-right corner. Click this icon to add that page to your Home page for quick "one-click" access in the future.

Summary

This simple `Hello World` procedure demonstrated how to quickly create a procedure and how any command can be wrapped and parameterized using a Commander procedure. If you have an existing script you want to automate, you can call that script within a step command block. This scenario also introduced you to running a job, entering parameters for the job, and viewing results for the job you ran.

Go to [Scenario 2](#)

Getting Started - Scenario 2 - Creating a procedure that uses an SCM

Scenario requirements: Before you begin this scenario, Commander needs to be able to access your Source Control Management (SCM) system.

To set up a Source Control system (SCM) to use with Commander, you need:

- a running SCM system
- an active source depot in your SCM system
- a Commander agent that can communicate with your SCM server

Note: If the machine where you installed Commander can communicate with your Source Control (SCM) server, this requirement is met. *If not*, return to the installation executable file and run the Commander Agent install program on a machine that has access to your SCM system.

To install Commander agent software on another machine, copy the installation executable file to that machine, then double-click on the file to run the installation program. Select Express Agent when the install window opens.

When installation is complete, configure this new agent as a Commander Resource. See the beginning of [Scenario 4](#) for more information.

- Configure Commander to communicate with your SCM system.
- Select the Administration tab, then the Source Control subtab.
- The Source Control Configurations page displays a table listing Source Control configurations after you create them. Click the **Create Configuration** link and the following web page is displayed.



- On the New Source Control Configuration web page, use the drop-down menu to choose your SCM type.
For our example, we chose **Perforce**.
- Depending on which SCM you choose, the page expands to display appropriate fields for supplying values to configure your SCM.

See the next screen example.

Fill-in the fields as follows:

- Configuration Name - For our example, we use the name, "p4". You can choose any name for your SCM Configuration.

Note: You will need this name later, so remember the name you choose for your SCM configuration. A configuration name is important because you may want to create more than one source control configuration.

- P4PORT - For our example, we use p4:3710 to designate the P4PORT. Your specification will be different.

Important: Other fields remain blank to use the defaults, but you may need to specify some or all information in the remaining fields depending on how your internal systems and access are setup for access to your SCM system.

Click **OK** after filling-in the fields.

For additional help with this Commander web page, click the **Help** link in the top-right corner of the web page.

Overview

This scenario guides you through creating a common build process integrated with your SCM system and build utility. And you will learn how to navigate and modify the procedure definition using the Commander Procedure Details web page.

At the end of this scenario, you will be familiar with the following Commander concepts and features:

- Source control configuration
- Creating more complex steps
- Creating a build process

- Navigating the Procedure Details web page

Begin Scenario 2

Step 1. Select the Projects tab

On the Projects page, select the project name you created in Scenario 1 (in the first column) to go to the Project Details page, then select the **Create Procedure** link to begin.

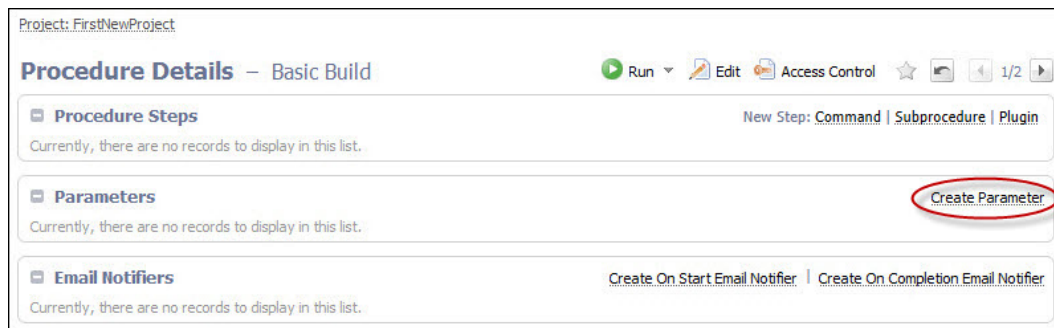
Step 2. Create a procedure

On the New Procedure page, supply a Procedure Name - it is **required**.

You can use any unique name you choose—**do not** use the same procedure name you chose for Scenario 1. Procedure names within the same project must be unique.

- Name - For this scenario, we use `Basic Build` for the procedure name.
- Default Resource - Use `local` for the resource name.

Click **OK** to go to the Procedure Details page.



Step 3. Create a parameter

On the Procedure Details page (illustrated above), click the **Create Parameter** link to go to the New Parameter page.

- Name - For our example, the parameter name is `target`. You can choose any unique name of your choice for your parameter name—parameter names must be unique within a procedure.
- Default Value - We chose `jar` for this value.

Click **OK** to continue and to return to the Procedure Details page to see the parameter you just created.

Step 4. Create a step to “checkout” source files from your SCM system

From the same Procedure Details page, click the Plugin link to create a New Step.

- In the Choose Step dialog, select Source Code Management from the left pane.
- Next, select ECSCM-Perforce from the list, then select "ECSCM-Perforce - Checkout" from the right pane.
- Click **Close** to go to the New Step page, which will be populated with the fields you need to fill-in for the Perforce SCM.
 - Both the Subprocedure and the Parameters sections are populated with fields to supply information so Commander can communicate with your Perforce SCM system, using Commander's "Perforce plugin".

Note: Depending on the SCM you chose to work with, including Perforce, the New Step screen you see may be different than the following example. If you need additional help configuring your SCM, see the corresponding SCM plugin on the Plugin Manager page—each plugin has its own Help topic.

New Step

General

Name: checkout

Description:

Subprocedure

Subprocedure: ECSCM-Perforce : CheckoutCode Change

Resources: Browse

Parameters

branch:

changelist:

checkoutSingleFile: ☐

client:

config: p4 Required

dest: project

forceSync: ☐

incremental: ☐

lastSnapshot:

template: lightning

updatesFile:

view:

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique name for your subprocedure step. For this scenario, the step name we supplied is <code>checkout</code> .
Description	(optional) Supply a text description for this step if you choose to do so.

Field Name	Description
<p>In the Subprocedure section:</p> <p>This section displays the subprocedure name or a plugin name because the plugin is being called as subprocedure.</p> <p>Note: The plugin name defined: "EC" is for Electric Cloud, "SCM" identifies the plugin as belonging to the Source Control Management category, and "Perforce" identifies the name of the source control system.</p> <p>Our example illustrates the fields for the Perforce SCM plugin. Your fields will be different if you are using a different SCM. These parameter fields request some of the same values you already specified when you created the Source Control Configuration.</p>	
ECSCM-Perforce	Clicking this link takes you to the Project Details page for the plugin. Generally, and for this scenario, you do not need to access this page to make changes—it is for your reference only.
CheckoutCode	Clicking this link takes you to the Procedure Details page for this procedure. Generally, and for this scenario, you do not need to access this page to make changes—it is for your reference only.
Change	Clicking this link displays a dialog that allows you to change the project, plugin, or subprocedure. If you make a change in the Change Subprocedure dialog, the Subprocedure and Parameters sections on the New Step page will update automatically with corresponding fields for your new choice. Note: All SCM plugins that can checkout code contain the <code>checkoutCode</code> procedure.
Resource	The <code>local</code> resource was specified earlier. If this field remains blank, the <code>local</code> resource is used.
Note: The following two parameter fields are common among all SCM plugins:	
config	The name of your Source Control Configuration.
dest	The destination where you want to put your checked-out code—it is the directory within your job workspace. Each job adds a new directory to the workspace. At this point, you have not defined multiple workspaces, so the Commander default workspace is used.

For more information about workspaces, see the "[Workspaces and Disk Space Management](#)" help topic.

Click **OK** after supplying values and to continue to the Procedure Details page, which should now look similar to the following example:

Procedure Details — Basic Build

Run Edit Access Control 1/2

Procedure Steps New Step: Command | Subprocedure | Plugin

Step Name	Resource	Action	Parallel	Time Limit	Actions
checkout		ECSCM-Perforce:CheckoutCode			Copy Delete

Parameters Create Parameter

Parameter Name	Default Value	Type	Req?	Description	Actions
target	jar	entry			Delete

The Procedure Details page now displays the name of our new procedure, Basic Build, and its new checkout step using the Perforce SCM, and the target parameter with the jar value.

Step 5. Create a step to run the build

This step can be another subprocedure step, a command step, or a custom step. Any procedure can contain multiple types of steps, any of which may be either simple or complex.

Steps can contain or use:

- parameters and/or properties
- many of your existing scripts, including shell scripts

For this step example (to run the build), we will create an Ant step. To invoke Ant, you could write a script that calls Ant directly, or you could call the Commander Ant plugin to invoke the runAnt procedure. For this example, we will use the Commander Ant plugin.

- From the Procedure Details page, click the **Plugin** link to see the Choose Step dialog and select the Build category.
- Select the EC-Ant plugin to see and select the "Ant - Run Ant" step.
- When the New Step page is displayed, notice the Subprocedure section is populated with the chosen EC-Ant plugin, and the Parameters section provides the appropriate fields for this Ant step.

New Step

General

Name: AntBuild

Description:

Subprocedure

Subprocedure: EC-Ant : runAnt Change

Resource: Browse

Parameters

Ant Location:

Build File: build.xml

Libraries:

Properties:

Property File:

Debug: ☐

Diagnostics: ☐

Output Level: ☒ Normal ☐ Quiet ☐ Verbose

Log File:

Target: \${target}

Additional Commands:

Postp Line:

Working Directory: project

Fill-in the fields as follows:

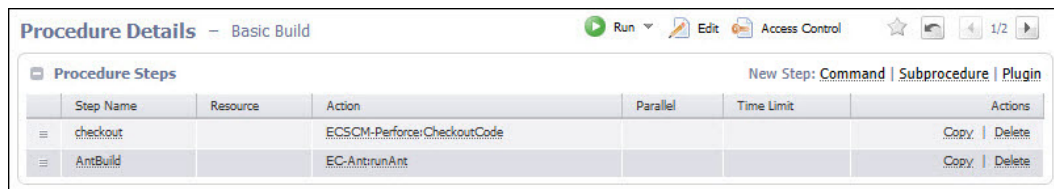
Field Name	Description
Name	For this example, we specified <code>AntBuild</code> for the step name. You can use any unique name you choose for this step.
Subprocedure section: Note the EC-Ant plugin project name and the <code>runAnt</code> procedure name.	
Resource	Leave this field blank to continue using the local resource.
Parameters section:	

Field Name	Description
Build File	Use <code>build.xml</code> in this field.
Target	For this example, we used <code>\${target}</code> .
Working Directory	For this example, we used <code>project</code> .

For this scenario, we leave the remaining fields blank, but you may need to supply additional information depending on your Ant invocation.

Click **OK** to continue and return to the Procedure Details page.

The following screen example illustrates the two steps we created for this scenario.

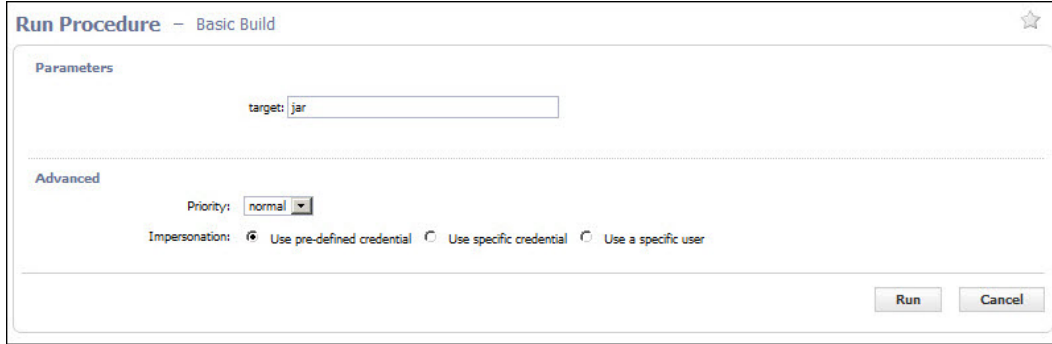


Step 6. Running the procedure

On the Procedure Details page, hover your mouse over the small down-arrow on the right-side of the **Run** link, choosing **Run...**

- **Run Immediately** - This option runs the procedure immediately as set.
- **Run...** - This option displays the Run Procedure web page where you may alter any existing parameters for this procedure, or set an existing credential.

Note: This scenario does not introduce you to the Advanced section of the Run Procedure page. For more information on these topics, click the **Help** link in the top-right corner of the Run Procedure page.



Run Procedure – Basic Build

Parameters

target:

Advanced

Priority:

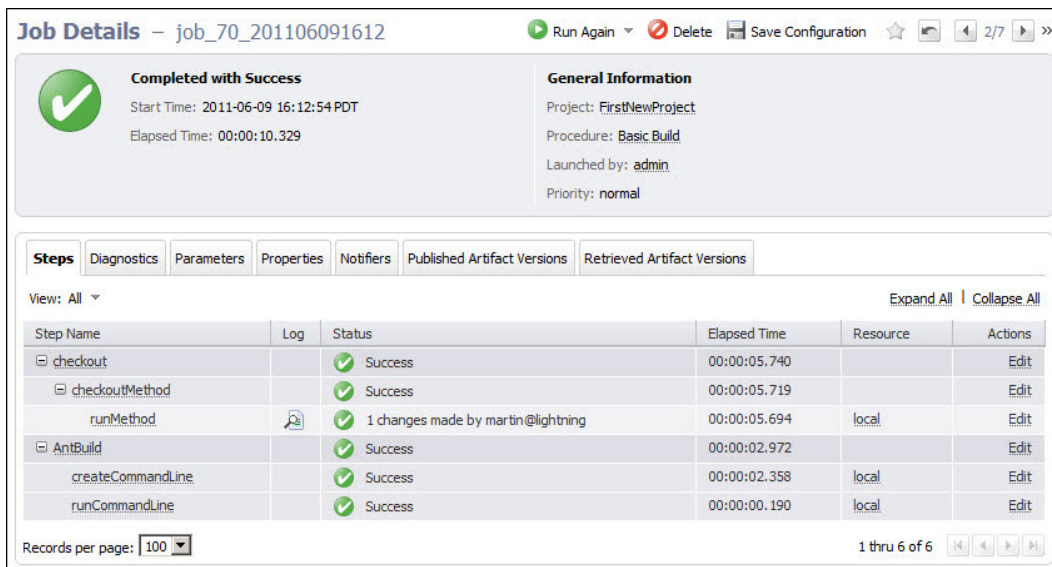
Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Click **Run** to run the procedure and continue to the Job Details page to see the status of your running job.

Step 7. On the Job Details page...

The Job Details page now displays the status and results of running the Basic Build procedure.

Your Job Details page will look somewhat different if you supplied other values in this scenario.



Job Details – job_70_201106091612

2/7

Completed with Success

Start Time: 2011-06-09 16:12:54 PDT
Elapsed Time: 00:00:10.329

General Information

Project: [FirstNewProject](#)
Procedure: [Basic Build](#)
Launched by: [admin](#)
Priority: [normal](#)

Steps | [Diagnostics](#) | [Parameters](#) | [Properties](#) | [Notifiers](#) | [Published Artifact Versions](#) | [Retrieved Artifact Versions](#)

View: All

Step Name	Log	Status	Elapsed Time	Resource	Actions
[-] checkout		✓ Success	00:00:05.740		Edit
[-] checkoutMethod		✓ Success	00:00:05.719		Edit
[-] runMethod		✓ 1 changes made by martin@lightning	00:00:05.694	local	Edit
[-] AntBuild		✓ Success	00:00:02.972		Edit
[-] createCommandLine		✓ Success	00:00:02.358	local	Edit
[-] runCommandLine		✓ Success	00:00:00.190	local	Edit

Records per page: 1 thru 6 of 6

More about the Job Details page

- The General Information section at the top of the table provides links back to the Project > Procedure that was launched to create this job.
- Click on a Step Name to go to the Job Step Details page.
- Click the Parameters tab to see parameters in this job.
- In case of errors, the Diagnostics tab provides specific information.

More about the Procedure Details page

To go to the Procedure Details page at any time:

- Select the Projects tab.
- Select the Project name (that contains the procedures you want to see) to go to the Project Details page.
- Select a Procedure name to go to that procedure's Procedure Details page.

You may want to review to the Procedure Details page frequently because it displays the steps that make up your procedure, the parameters contained in the procedure, email notifiers, and more. Full edit capabilities are available by selecting any object (step, parameter, and so on).

Also, you can run this procedure again “on demand” by clicking the **Run** link at the top of the page.

For more information about the Procedure Details page functions, click the **Help** link in the top-right corner of the Procedure Details page.

Scenario extension - Add a step to show workspace contents

If you are curious about what your current default workspace contains at this point, you can create a step to see the workspace.

- Select the Projects tab, select FirstNewProject (or your project name if you used something different), select the Basic Build procedure name.
- Click the **Command** link to create a New Step—a new *command* step.
- The New Step page is displayed with a Command text box.
 - For Name, use `seeWorkspace`.
 - In the Command(s) text box:
 - for Windows - type `dir /s`
 - for Linux - type `ls -r`
- Click **OK** to create the step and go to the Procedure Details page.
- Click **Run** to create the job and to go to the Job Details page.

When the job is completed, click the log icon in the Log column to see the contents of the current workspace for this procedure. Your Workspace File will be similar to the following example:

Job: job_198_201106140943

Workspace File – seeWorkspace.1375.log

Volume in drive N has no label.
Volume Serial Number is EAB8-74CE

Directory of N:\job_198_201106140943

```
06/14/2011  09:43 AM    <DIR>        .
06/14/2011  09:43 AM    <DIR>        ..
06/14/2011  09:43 AM    <DIR>        project
06/14/2011  09:43 AM                2,770 runMethod-checkoutCode-1377.log
06/14/2011  09:43 AM                0 seeWorkspace.1375.log
                2 File(s)                2,770 bytes
```

Directory of N:\job_198_201106140943\project

```
06/14/2011  09:43 AM    <DIR>        .
06/14/2011  09:43 AM    <DIR>        ..
06/14/2011  09:43 AM                1,141 build.xml
06/14/2011  09:43 AM                338 Makefile
06/14/2011  09:43 AM                338 Makefile2
06/14/2011  09:43 AM    <DIR>        src
                3 File(s)                1,817 bytes
```

Directory of N:\job_198_201106140943\project\src

```
06/14/2011  09:43 AM    <DIR>        .
06/14/2011  09:43 AM    <DIR>        ..
06/14/2011  09:43 AM                144 HelloWorld.java
                1 File(s)                144 bytes
```

Total Files Listed:

```
6 File(s)                4,731 bytes
8 Dir(s)  244,679,413,760 bytes free
```

Summary

This scenario demonstrated how easily Commander can integrate with your SCM system and build utilities.

Go to [Scenario 3](#)

Getting Started - Scenario 3 - Notification, scheduling, and reporting

Scenario requirements: For this scenario, you need to set up an Email Configuration to use the email notification feature.

To set up an Email Configuration:

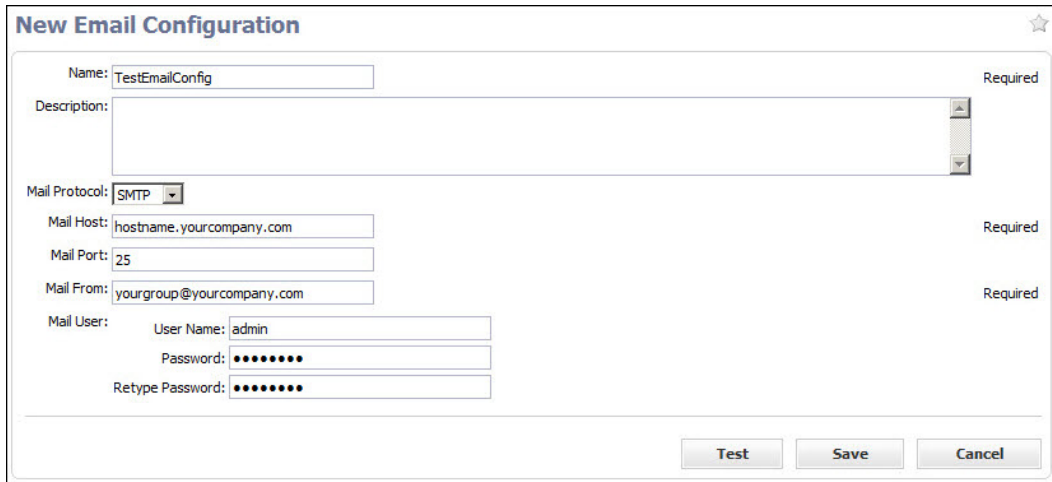
- Select the Administration tab, then select the Email Configurations subtab.
You will see a blank Email Configurations page. As you create one or more email configurations, they will be listed in table format on this page.
- Click the **Add Configuration** link.

On the New Email Configuration page, fill-in the fields to define your mail system.

Field Name	Description
Name	Supply a unique name for your email configuration.
Description	(optional) Supply a text description for your reference.
Mail Protocol	Use the drop-down arrow to make a selection.
Mail Host	Supply the name of your mail host.

Fill-in the remaining fields according to the specifications you need.

Your New Email Configuration page should look similar to the following example:



The image shows a 'New Email Configuration' dialog box. It has a title bar with a star icon. The form contains the following fields and controls:

- Name:** A text box containing 'TestEmailConfig'. To its right is a 'Required' label.
- Description:** A large text area with a vertical scrollbar.
- Mail Protocol:** A dropdown menu currently showing 'SMTP'.
- Mail Host:** A text box containing 'hostname.yourcompany.com'. To its right is a 'Required' label.
- Mail Port:** A text box containing '25'.
- Mail From:** A text box containing 'yourgroup@yourcompany.com'. To its right is a 'Required' label.
- Mail User:** A section containing three sub-fields:
 - User Name:** A text box containing 'admin'.
 - Password:** A text box filled with dots.
 - Retype Password:** A text box filled with dots.

At the bottom right of the dialog are three buttons: 'Test', 'Save', and 'Cancel'.

Click **Save** after filling-in the fields or click the **Test** button first if you want to make sure your email configuration works.

Supply an email address in the pop-up box and click **Send**.

Note: For additional help with this Commander page, click the **Help** link in the top-right corner of the page.

Overview

This scenario guides you through creating an email notification, invoking your build process with a timed (standard) schedule, invoking your build process using a continuous build integration trigger, and introduces Commander reporting features.

At the end of this scenario, you will be familiar with the following Commander concepts and features:

- Creating an email configuration
- Creating an email notifier
- Creating schedules
- Creating reports

Begin Scenario 3

Step 1. Creating an email notifier

On the Procedure Details page for the Basic Build procedure, click the **Create On Completion Email Notifier** link to begin. This link takes you to the New On Completion Email Notifier page.

Procedure Details – Basic Build

Run Edit Access Control 1/2

Procedure Steps New Step: Command | Subprocedure | Plugin

Step Name	Resource	Action	Parallel	Time Limit	Actions
checkout		ECSCM-Perforce-1.1.11.40415:CheckoutCode			Copy Delete
AntBuild		EC-Ant-1.0.3.40250:runAnt			Copy Delete
seeWorkspace		dir /s			Copy Delete

Parameters Create Parameter

Parameter Name	Default Value	Type	Req?	Description	Actions
target	jar	entry			Delete

Email Notifiers Create On Start Email Notifier Create On Completion Email Notifier

Currently, there are no records to display in this list.

Two email notifier types for jobs or job steps:

- On Start Notifier - sends an email when a job or job step starts.
- On Completion Notifier - sends an email when a job or job step completes.

The fields for each of these notifiers are similar, but we chose the On Completion notifier because you may find you use this one more frequently.

On the New On Completion Email Notifier page:

Fill-in the fields as follows:

Field Name	Description
Name	For this example, we use <code>BuildTeam</code> . This name can be an arbitrary text string.
Description	Supply a text description for your reference if you choose—Commander does not use or interpret this information.
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty, non-zero, or "true", the message is sent. If the result is "0" or "false", the message is not sent.

Field Name	Description
Formatting Template	<p>For this example, choose the "Job summary HTML" notification template, or you can use the drop-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different.</p> <p>To customize your template, edit the auto-supplied text in the Formatting Template text box or you can add a completely new template for your purpose. Note: Any edits made in this text box will not be saved to the global template, and the template will appear as a Custom template when you return to this notifier definition. Additionally, make sure the content is formatted correctly, i.e., no illegal characters or spacing.</p>
Email Configuration	Use TestEmailConfig, the email configuration we created.
Destinations	This is a space-separated list of valid email addresses, email aliases, or ElectricCommander user names, or a property reference that expands into such a list.

After filling-in the fields, your page may look similar to the following example:

New On Completion Email Notifier

Name: BuildTeam Required

Description:

Condition: On error

`[/javascript if(getProperty("outcome") == 'error')
true;
else
false;]`

Formatting Template: Job summary HTML notification

Subject: Job '[jobName]' from procedure '[procedureName]' [/myEvent/type] - Commander notification Required

`[/server/ec_notifierTemplates/Html_JobTempl/body]`

Email Configuration: TestEmailConfig

Destinations: john@yourCompany scott@yourCompany Required

OK Cancel

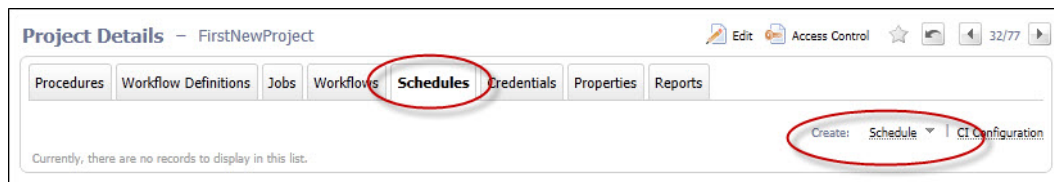
Click **OK** to create your email notifier and to return to the Procedure Details page, where you can now see the Email Notifier you created.

Note: For more information, see the [Email Notifiers](#) help topic.

Step 2. Creating a standard schedule

To create a standard "timed" schedule:

- Select the Projects tab.
- Select your project name to go to the Project Details page.
- Select the Schedules subtab, then select the down-arrow to the right of the "Schedule" link, and choose "Standard Schedule" to go to the New Schedule page.



On the New Schedule page:

The following is an example of the New Schedule page.

- Notice the "breadcrumb" above the New Schedule page title—this entry displays the name of the project that will contain this schedule.
- After supplying a name (unique within your project) for your schedule (we chose "Sentry-MainBuild"), choose the procedure your schedule will run.
 - Click the **Change** link adjacent to the Procedure field.
 - In the popup menu, leave the Current project selected.
 - Place your cursor in the Procedure field to see a list of procedures in your project.
 - Select the procedure you want this schedule to run. We chose "Basic Build".

Project: FirstNewProject

New Schedule

☆

General

Name:

Procedure: FirstNewProject : Basic_Build [Change](#)

Description:

Parameters

target:

Frequency

Run at 00:00.

Days: ☒ Every Day ☐ Once ☐ Days of Week ☐ Days of Month ☐ Custom

Repetition: Run Once at (h:mm)

Advanced

Enabled: ☒

Misfire Policy: Ignore the missed triggers and wait for the next scheduled time

Time Zone: [- Default -]

Priority: normal

Impersonation Credential

Credential Project: ☒ Current ☐ [Browse](#)

Credential Name: [Browse](#)

OK

Cancel

Fill-in the remaining fields as follows:

Field Name	Description
Description	(optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Parameters section	
If the procedure has parameters, fill-in their values in this section. The procedure cannot execute unless all required parameters are provided.	
Frequency section	
"Run at" is a summary section: As you make selections for the following Days and Repetition sections, a summary of what you selected appears here.	

Field Name	Description
Days	Select weekdays, month days, or create a custom frequency when you want the schedule to run. Note: The "day" applies to the start time if the time range spans two days, that is, if the time range crosses the midnight boundary.
Repetition	If Run Once - the schedule will run once per selected day, at the time you specify. If Run Every - the procedure will execute repeatedly during the specified time range. For example, if you select a time range from 11:00-14:00 and an interval of 40 minutes, the procedure will execute 5 times on each of the selected days: at 11:00 am, 11:40 am, 12:20 pm, 1:00 pm, and 1:40 pm. If Run Continuously - as soon as one job ends, the scheduler will trigger immediately to run the job again.
Advanced section	
Enabled	If this box is "checked," the schedule will run. You can disable this schedule whenever necessary.

Note: For this scenario, you do not need to supply information for any remaining fields. Later, when creating an actual timed schedule for one of your procedures, refer to the help topic for this page for more information.

Click **OK** to save your information and create this schedule.

At any time, you can return to the Project Details page for this project, select the Schedules subtab, then select the Schedule Name from the table to go to the Edit Schedule page to modify this schedule.

Step 2a. Creating a continuous integration build trigger

At a later time when you have procedures and steps created to do the work you need, you may prefer implementing continuous integration build configurations for your software builds. Unlike a standard schedule that runs on days and times you specify, a continuous integration build can be triggered to run each time code is checked into your source code management (SCM) system.

Adding a project to the Continuous Integration Dashboard is quick and easy. Each project can have one or more CI configurations, depending on the number of source branches you want to monitor. After the simple configuration process, you can visually see status and more for your continuous integration jobs as builds are triggered to run.

To find the Continuous Integration Dashboard, select the Home tab, then select the Continuous Integration subtab. The dashboard will be empty until you start adding projects. For help using the Continuous Integration

Manager and Dashboard, click the **Help** link in the upper-right corner of the dashboard web page.

Step 3. Creating a report

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

Summary of available report types:

- Real-time reports - filtered view of your Commander data in real-time
- Build reports - summary reports produced at the end of a build and attached to the job
- Batch reports - summaries of your build environment with trends over time, two types:
 - Default Batch reports - automatically installed during ElectricCommander installation and scheduled to run daily (Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)
 - Optional Batch reports - you can configure and schedule these reports to fit your requirements (Category, Procedure Usage, Count Over Time, or Multiple Series reports)
- Custom reports - your choice to create and add at any time

Note: Batch and Custom reports must be run on the Commander server's agent (local agent) only. These reports use the BIRT report engine, which is installed only on the Commander server.

Defining a “saved” search

Before you create a report, you need to define a "search" to focus the report on the information you want to see.

- Select the Search tab to go to the Define Search page.

Fill-in the fields as follows:

Field Name	Description
Number of results	For this example, do not alter the default settings in these fields as you see them. These fields do not affect the search you are creating.
Results Per Page	
Object Type	For this example, choose Job for this search.
Intrinsic Filters section	Click the Add Intrinsic Filter link one or more times as needed to further define the sort criteria for your search. We selected Project Name from the drop-down menu and typed-in our project name, then selected Procedure Name and typed-in our <code>Basic Build</code> procedure name.
Custom Filters section	Not needed for this scenario.

Your Define Search page will look similar to the following example:

Define Search

General

Number of Results:

Results Per Page:

Object Type:

Intrinsic Filters

equals [Remove](#)

equals [Remove](#)

[+ Add Intrinsic Filter](#)

Custom Filters

[+ Add Custom Filter](#)

[OK](#) [Cancel](#)

Click **OK** after filling-in the fields and to go to the Search Results page.

Job Search Results

9 Results for "Project Name" equals "FirstNewProject" and "Procedure Name" equals "Basic Build"

[New Search](#) [Save Filter](#) [Edit Search](#) [Refresh Search](#)

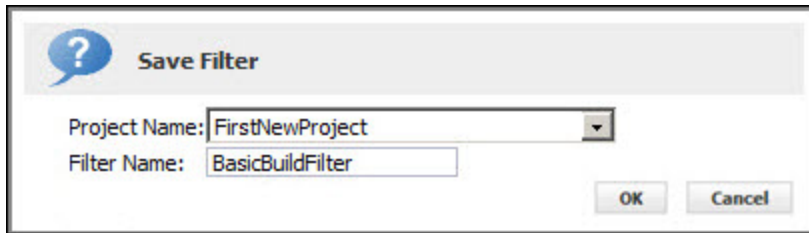
Job	Status	Priority	Procedure	Launched By	Elapsed Time	Start Time	Actions
job_1250_201104121322	Success	normal	FirstNewProject:Basic Build	admin	00:00:14.124	2011-04-12 13:22:11 PDT	Delete
job_1249_201104121321	Success	normal	FirstNewProject:Basic Build	admin	00:00:12.260	2011-04-12 13:21:56 PDT	Delete
job_1248_201104121321	Success	normal	FirstNewProject:Basic Build	admin	00:00:12.297	2011-04-12 13:21:37 PDT	Delete
job_1247_201104121321	Success	normal	FirstNewProject:Basic Build	admin	00:00:13.525	2011-04-12 13:21:17 PDT	Delete
job_1209_201104121015	Success	highest	FirstNewProject:Basic Build	project: Electric Cloud	00:00:14.965	2011-04-12 10:15:19 PDT	Delete
job_1086_201104081645	Success	normal	FirstNewProject:Basic Build	admin	00:00:18.129	2011-04-08 16:45:14 PDT	Delete
job_1084_201104081644	Success	normal	FirstNewProject:Basic Build	admin	00:00:14.537	2011-04-08 16:44:24 PDT	Delete
job_1079_201104081623	Success	normal	FirstNewProject:Basic Build	admin	00:00:14.499	2011-04-08 16:23:25 PDT	Delete
job_806_201104060942	Success	normal	FirstNewProject:Basic Build	admin	00:00:12.045	2011-04-06 09:42:49 PDT	Delete

Records per page:

1 thru 9 of 9

Click **SaveFilter** to see the Save Filter dialog box.

In this dialog box, use the drop-down arrow to select the project name for your report, then supply a name of your choice for the filter. We used `BasicBuildFilter` for the filter name to tie the filter name to the Basic Build procedure for which we are creating a filter.



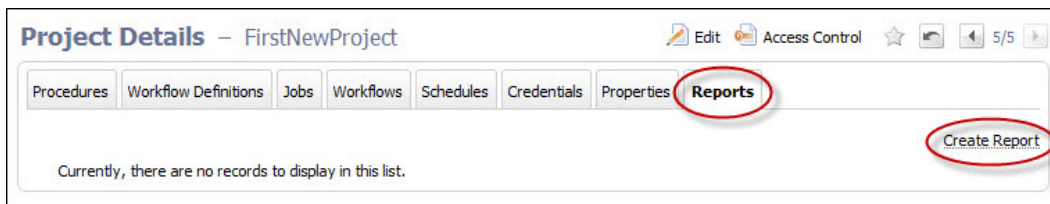
A dialog box titled "Save Filter" with a question mark icon. It contains two input fields: "Project Name:" with a dropdown menu showing "FirstNewProject", and "Filter Name:" with a text box containing "BasicBuildFilter". At the bottom right are "OK" and "Cancel" buttons.

Click **OK** to return to the Job Search Results page—now we are ready to define the report we want to see.

Define the report

For this scenario, we will create an Optional Batch report. Select the Projects tab, select *your* project name to go to the Project Details page, then select the Reports subtab.

Click the **Create Report** link.



The "Project Details" page for "FirstNewProject". It features a navigation bar with tabs: Procedures, Workflow Definitions, Jobs, Workflows, Schedules, Credentials, Properties, and Reports. The Reports tab is highlighted with a red circle. Below the tabs, a message states "Currently, there are no records to display in this list." In the bottom right corner, a "Create Report" link is also highlighted with a red circle.

On the Optional Batch Reports page:

For this scenario, select the Multiple Series report tab.

Fill-in the fields or select the appropriate values as follows:

Field Name	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. We supplied <code>Basic Build Report</code> for our report name.
Saved Filter	Project - Click your mouse inside this field to see a list of projects from which to make a selection. We select our <code>FirstNewProject</code> project name. Filter - Use the <code>BasicBuildFilter</code> we created earlier.
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box so you can view this report on your Home page. (We will create a "thumbnail" report view on your Home page at the end of this scenario.)

Field Name	Description
Object Type	Use the drop-down menu to select Job for this example.
Table column choices	
Chart Type	Use the drop-down menu to choose the chart type.
Function	Use the drop-down menu to choose the function you need.
Property Name	Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. We chose "outcome" for this example.
Display Name	You can choose a different unique Display Name if you prefer to do so.
Stacked	Select this checkbox to see your report results "stacked" versus overlaid.
The "X" icon	Click this icon to delete any row you no longer need.
Add Series button	Select this button if you would like to create additional table entries for additional report information.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.

Your Multiple Series report page will be similar to the following example:

Multiple Series | Category | Count Over Time | Procedure Usage

Report Title: Basic Build Report

Saved Filter: Project: FirstNewProject
Filter: BasicBuildFilter

Time Period: This Week

Create thumbnail? ☒

Object Type: Job

Chart Type	Function	Property Name	Display Name	Stacked
Bar	Sum	outcome	Outcome	<input checked="" type="checkbox"/>

Add Series

▼ Chart Options

Time Grouping: Auto

X-Axis Label: Date

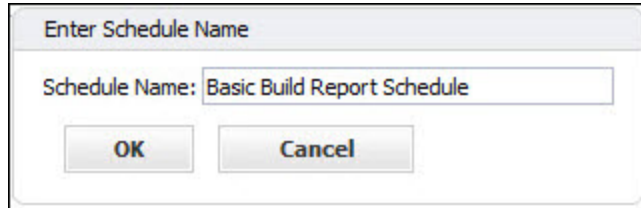
Y-Axis Label: Number of jobs

► Table Options

► Advanced

Run Report | Create Schedule | Cancel

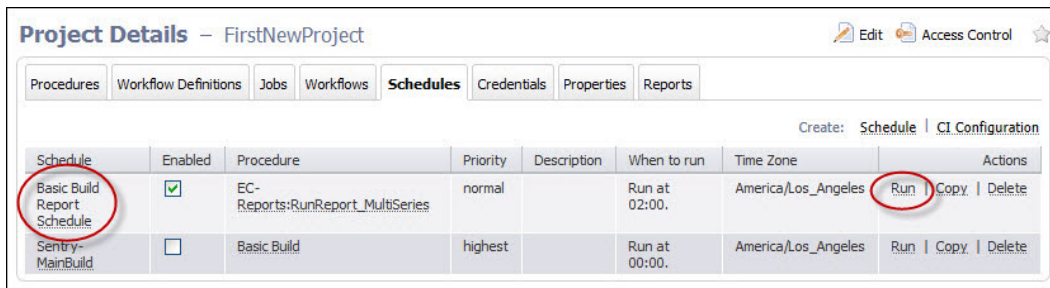
- Run Report button - (information only for this scenario) this button takes you to the Job Details page to run the report immediately—one time only.
- Click the **Create Schedule** button - this button displays a box with a default schedule name you can change.



Enter Schedule Name

Schedule Name:

- Supply a name for the schedule—again, we tied the schedule name to the procedure name for which we want the report.
- Click **OK** to go to the Project Details page.



Project Details – FirstNewProject

Procedures Workflow Definitions Jobs Workflows **Schedules** Credentials Properties Reports

Create: [Schedule](#) | [CI Configuration](#)

Schedule	Enabled	Procedure	Priority	Description	When to run	Time Zone	Actions
Basic Build Report Schedule	<input checked="" type="checkbox"/>	EC-Reports:RunReport_MultiSeries	normal		Run at 02:00.	America/Los_Angeles	Run Copy Delete
Sentry-MainBuild	<input type="checkbox"/>	Basic Build	highest		Run at 00:00.	America/Los_Angeles	Run Copy Delete

Now you can see the report you just created listed in the Schedules table.

Click the **Run** link in the Actions column to run this report and then go to the Job Details page.

On the Job Details page, you will see your report running.

When the report is generated, you will see a Links section in the summary section at the top of the Job Details page.

This section contains a link to the report you generated.

Click this link to see your report.

Job Details – EC-Reports-1.1.3.36678-RunReport_MultiSeries-1281

[Run Again](#) [Delete](#) [Star](#) [More](#)

Completed with Success

Start Time: 2011-04-12 15:46:08 PDT
Elapsed Time: 00:00:48.979

General Information

Project: FirstNewProject
Procedure: EC-Reports-1.1.3.36678:RunReport_MultiSeries
Scheduler: Basic Build Report Schedule
Launched by: admin
Priority: normal

Links

[Basic Build Report](#)

Steps | Diagnostics | Parameters | Properties | Notifiers

View: All [Expand All](#) | [Collapse All](#)

Step Name	Log	Status	Elapsed Time	Resource	Actions
RunCustomReport	Log	Success	00:00:47.387	local	Edit

For additional information about the Job Details page, click [here](#).

For more information about Commander reports, see the [Reports](#), [Creating Custom Reports](#), and other "report" Help topics.

Scenario extension - Add a thumbnail report to your Home page

When we created a report, we "checked" the Create thumbnail? checkbox, so we are ready to activate the thumbnail report view on the Home page.

Note: Viewing an interesting report assumes there is ample data to report. While you can see a report after running one job, it is not very interesting unless you can see trends or comparisons after running numerous jobs.

To see a thumbnail report on the Home page:

- Select the Home tab.
- Click the **Add Report** link and a New Report page is displayed.

Home | Projects | Jobs | Workflows | Cloud | Artifacts | Search | Administration

Job Configurations [Create](#)

Currently, there are no records to display in this list.

Shortcuts [Create](#)

Currently, there are no shortcuts to display in this list.

Jobs Quick View

[Add Category](#) [Modify/Delete](#)

Last 10 Jobs

Job Name	Status	Elapsed Time	Details
Electric Cloud-ElectricSentry-1924	Success	00:00:09.628	Details
EC-Reports-1.2.0.0-RunReport_MultiSeries-1824	Success	00:00:55.182	Details
EC-Reports-1.2.0.0-RunReport_CategoryPie-1823	Success	00:00:57.255	Details

Reports

[Add Report](#)

Recent Job Outcome [Rename](#) | [Remove](#)

On the New Reports page:

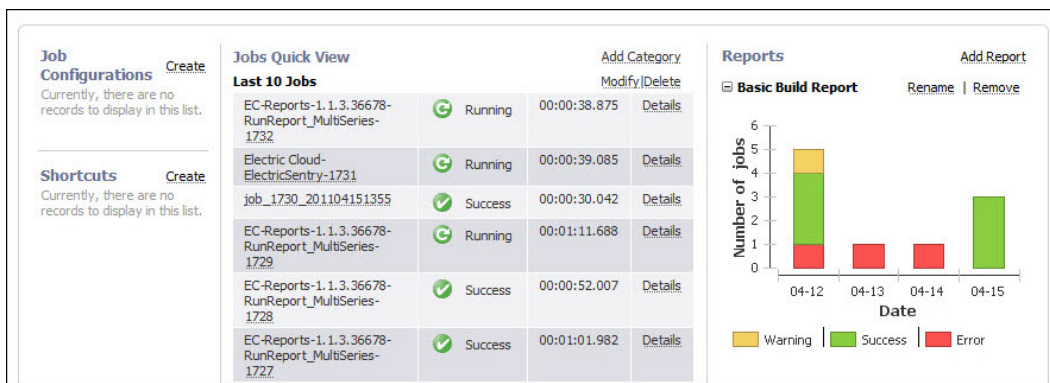
New Report

Project:	FirstNewProject	Required
Report:	Basic Build Report	Required
Title:	Basic Build Report	Required

- Project - Use the down-arrow to select the project containing the report you want to view—FirstNewProject.
- Report - This is the report name you entered in the Report Title field when you configured this report—Basic Build Report.
- Title - This is the report title you want to see on your Home page to identify this report. We used the same Basic Build Report name.

Click **OK** to generate the thumbnail report view and to go to the Home page.

The following screen example illustrates how your Home page may look with the thumbnail report.



For more information about how you can use the Home page, see [The Home Page](#) help topic.

Summary

This scenario demonstrated how to create an Email Configuration so you could use an email notifier, how to create a continuous integration schedule, and how to set up a basic report with a "thumbnail" report view on your Home page.

Go to [Scenario 4](#)

Getting Started - Scenario 4 - Multi-agent build and test

Scenario requirements: For this scenario, you can use the configurations you created for Scenario 2 and 3. However, before you begin this scenario, you need to configure an additional agent machine. If you have a firewall running on the Commander server machine, disable it now or allow access to ports 8000, 8443, 61613, 80, and 443.

Installing Commander Agent software:

Install Commander Agent software on the machine you intend to use as an agent to run builds. For the simple purpose of getting started, install this agent on a different machine with the same operating system as the Commander server.

- Copy the ElectricCommander installation executable file to the machine you intend to use as a Commander agent.
- Double-click the executable file to begin the installation program.
- When the install program opens, select the Express Agent installation.

Note: During the agent installation, you will have the opportunity to create a resource and provide a resource name. Supply any name you choose for your new resource. For this scenario, we will name our new resource, `Resource2`.

- After agent installation, verify that your new resource information is included in the Commander resource table.
 - Select the Cloud tab to go to the Resources page.

Resources -- Current License Usage: 0 of unlimited Managed Hosts; 0 of unlimited Proxied Hosts

Filters

filter1 X

Save Filters Reset

Quick Search

Status

Both Enabled/Disabled

Pools

		Name	Pools	Job Status	Description	Zone	HTTPS & Host	Step Load
<input type="checkbox"/>		local	default		Local resource created during installation.	default	rh664r12.electric-cloud.com	0 of unlimited
<input type="checkbox"/>		local copy	default		Local resource created during installation.	default	rh664r12.electric-cloud.com	<input type="text"/> 0 of 3

Note: At a later time when you are more familiar with Commander, you can use the Resources page to install additional agents. See the Resources page Help topic, then specifically review the "Install or Upgrade Remotes Agents" section. Currently, with Commander v4.2, this new functionality available on the Resources page is available only for Linux agents.

To configure a shared workspace:

For Linux

If your agents are installed on Linux, change the default workspace to point to a network location accessible to both machines:

- Select the Cloud tab, then select the Workspaces subtab, to see a list of workspaces.
- Select the workspace named “default” to edit the default workspace.
- Change the value of the UNIX Path to a network location accessible to the Commander server and agent machines.
- Make sure this location is readable and writable by the user(s) chosen during the server and agent installations.

After fulfilling these scenario prerequisites, you are ready to proceed.

For Windows

If using a Windows environment, the shared workspace is created for you already.

Overview

This scenario guides you through creating a multi-agent build and test process integrated with your SCM system, build utility, and unit/system tests. You can invoke this build using a schedule, Continuous Integration (CI), or "on demand". At the end of this scenario you will be familiar with the following ElectricCommander concepts and features:

- Agent-only installation
- Copying a procedure to create a new procedure
- Running steps in parallel
- Pools
- Post processor
- Search

Begin Scenario 4

Step 1. Create a new procedure

This time, we are going to *copy* an existing procedure, rename it, and add another step.

- Select the Projects tab, then select the FirstNewProject project name.
- On the Project Details page, click the **Copy** link to copy the Basic Build procedure.

Project Details – FirstNewProject [Edit](#) [Access Control](#) [Star](#) [Refresh](#) [5/5](#)

Procedures Workflow Definitions Jobs Workflows Schedules Credentials Properties Reports

[Create Procedure](#)

Procedure ..	Description	Resource	Create Date	Actions
Basic Build		local	2011-03-25 14:32:14 PDT	Run ▾ Edit Copy Delete
Hello World	my first procedure	local	2011-03-25 10:36:38 PDT	Run ▾ Edit Copy Delete

Records per page: 20 1 thru 2 of 2

- When the Basic Build copy procedure is displayed, you can click its name to go to the Procedure Details page to verify it contains the same contents of the Basic Build procedure.
- Return to the Project Details page.

Project Details – FirstNewProject [Edit](#) [Access Control](#) [Star](#) [Refresh](#) [5/5](#)

Procedures Workflow Definitions Jobs Workflows Schedules Credentials Properties Reports

[Create Procedure](#)

Procedure ..	Description	Resource	Create Date	Actions
Basic Build		local	2011-03-25 14:32:14 PDT	Run ▾ Edit Copy Delete
Basic Build copy		local	2011-03-25 14:32:14 PDT	Run ▾ Edit Copy Delete
Hello World	my first procedure	local	2011-03-25 10:36:38 PDT	Run ▾ Edit Copy Delete

Records per page: 20 1 thru 3 of 3

- Select the **Edit** link for the Basic Build copy procedure.

On the Edit Procedure page:

- Name - supply a new procedure name, `MultiAgentBuild`.
- Description - Add a description noting this procedure was copied from the Basic Build procedure.
- Default Resource - Select your new resource, `Resource2`, for this procedure.

Click **OK** after supplying your information and Commander will take you back to the Project Details page.

Your Project Details page should now look similar to the following example:

Project Details – FirstNewProject [Edit](#) [Access Control](#) [Star](#) [Refresh](#) [5/5](#)

Procedures Workflow Definitions Jobs Workflows Schedules Credentials Properties Reports

[Create Procedure](#)

Procedure ..	Description	Resource	Create Date	Actions
Basic Build		local	2011-03-25 14:32:14 PDT	Run ▾ Edit Copy Delete
Hello World	my first procedure	local	2011-03-25 10:36:38 PDT	Run ▾ Edit Copy Delete
MultiAgentBuild	copied from the Basic Build procedure	Resource2	2011-03-25 14:32:14 PDT	Run ▾ Edit Copy Delete

Records per page: 20 1 thru 3 of 3

This page now displays the new MultiAgentBuild procedure with a description, and a new resource is designated for this procedure to use.

Step 2. Create two new Ant steps

- Beginning on the Project Details page, click the MultiAgentBuild procedure name to go to the Procedure Details page.
- Click the AntBuild step **Copy** link, twice.

Procedure Details – MultiAgentBuild						
Run Edit Access Control Star Refresh 3/3						
Procedure Steps						New Step: Command Subprocedure Plugin
	Step Name	Resource	Action	Parallel	Time Limit	Actions
≡	checkout		ECSCM-Perforce-1.1.11.40415:CheckoutCode			Copy Delete
≡	AntBuild		EC-Ant-1.0.3.40250:runAnt			Copy Delete
≡	AntBuild copy 2		EC-Ant-1.0.3.40250:runAnt			Copy Delete
≡	AntBuild copy		EC-Ant-1.0.3.40250:runAnt			Copy Delete
≡	seeWorkspace		dir /s			Copy Delete

- Now you see two additional steps names: AntBuild copy and AntBuild copy 2.
- Click the AntBuild copy step name to go to the Edit Step page.
- Notice the AntBuild copy step name is adjacent to the page title because this is the step we are going to edit.

Modifications for creating the new Ant steps:

Because this is an "Edit" page, some of the fields already contain the information we originally supplied to create the first Ant step. The following list contains the fields we need to edit or add new information:

- Name - For this example, we chose `unittest` for the step name. Type over the existing text to edit the text.
- Resource - Supply the name, `TestPool`, because we will be creating a resource pool for testing. At this point, you need to supply a pool name only (no spaces in the name).
- target - Edit this field to: `unittest`

In the Advanced section:

- Run in Parallel - Click the box to add a "checkmark". We are designating we want this step to run in parallel with one or more other steps.

Click **OK** to create the first of two new Ant steps and Commander returns you to the Procedure Details page.

Edit Step – AntBuild copy

Access Control

General

Name: unittest

Description:

Subprocedure

Subprocedure: EC-Ant : runAnt

Change

Resource: TestPool

Parameters

Ant Location: ant

Build File: build.xml

Libraries:

Properties:

Property File:

Debug: ☐

Diagnostics: ☐

Output Level: ☐ Normal
☐ Quiet
☐ Verbose
☒ normal

Log File:

Target: unittest

Additional Commands:

Postp Line:

Working Directory: project

Environment Variables:

Advanced

Precondition:

Run Conditions:

Error Handling: Procedure continues, but overall status will be error

Time Limit: minutes

Run in Parallel: ☒

From the Procedure Details page, click the AntBuild copy 2 step name to go to the Edit Step page again.

To create the next Ant step, make the same edits (above) as you did to create the `unittest` step, except:

- Name - Change the step name to `systemtest`.
- target - Change this field to: `systemtest`

Click **OK** to create the second of two new Ant steps and to return to the Procedure Details page—you now have 3 Ant steps, including 2 new Ant steps, `unittest` and `systemtest`, with `TestPool` as the designated resource.

Step Name	Resource	Action	Parallel	Time Limit	Actions
checkout		ECSCM-Perforce-1.1.11.40415:CheckoutCode			Copy Delete
AntBuild		EC-Ant-1.0.3.40250:runAnt			Copy Delete
unittest	TestPool	EC-Ant-1.0.3.40250:runAnt	✓		Copy Delete
systemtest	TestPool	EC-Ant-1.0.3.40250:runAnt	✓		Copy Delete
seeWorkspace		dir /s			Copy Delete

Also notice, the Parallel column now contains "checkmarks" for the 2 steps we want to run in parallel.

Step 3. Create a resource pool

Select the Cloud tab to see the following Resources page with a table listing the currently available resources.

Name	Pools	Job Status	Description	Zone	HTTPS & Host	Step Load
local	default	✓	Local resource created during installation.	default	rh664n2.electric-cloud.com	0 of unlimited
Resource2	default	✓	Local resource created during installation.	default	rh664n2.electric-cloud.com	0 of 3

To put these two resources in a resource pool that we previously "named" when we created the new Ant steps, we need to edit each resource.

Click the `local` resource name to go to the Resource Details panel, then click the **Edit** link at the top of the panel.

On the Edit Resource panel:

The only change we need to make is to add the `TestPool` pool name to the Pool(s) field.

Edit Resource

Name:

Description:

Agent Host Name:

Agent Port Number:

Default Workspace:

Pool(s):

Default Shell:

Step Limit:

Artifact Cache Directory:

Zone:

Repository Names:

Connection Type:

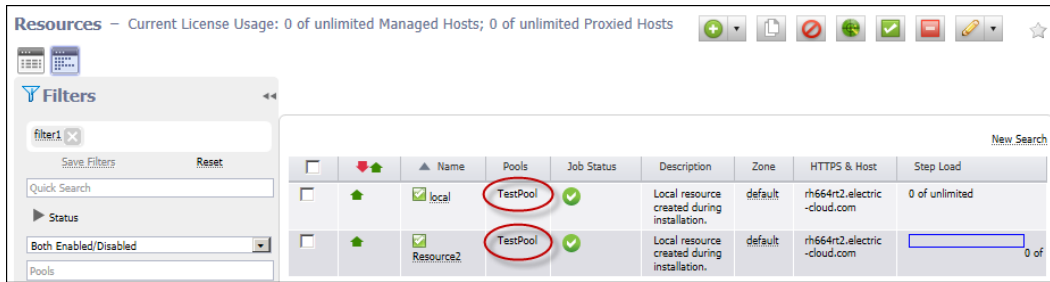
Enabled: ☒

Click **OK** to save your change and return to the Resources page.

- Notice the Pool(s) column now has TestPool specified as a resource pool where the `local` resource is a member.
- Repeat this process to put `Resource2` in the TestPool resource pool.

A resource can belong to one or more pools. If any resource belongs to multiple pools, all pools would be listed in this column. For more information about the Resources page, click [here](#) or click the **Help** link in the top-right corner of the web page.

The following screen example should be similar to your Resources page.



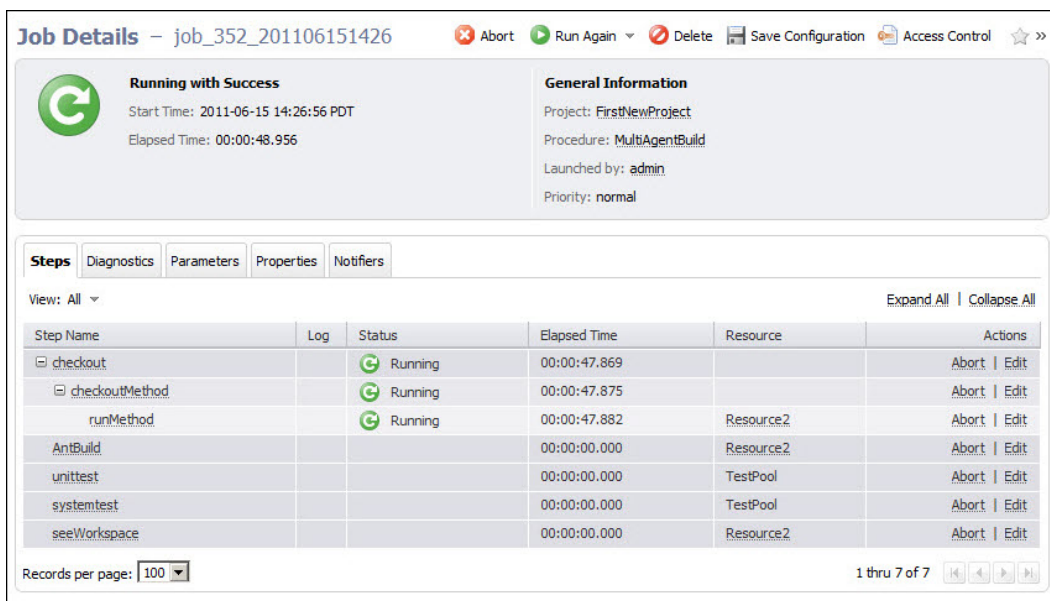
Step 4. Run the procedure

To run the new MultiAgentBuild procedure, select the Projects tab > FirstNewProject project name > MultiAgentBuild procedure.

On the Procedure Details page:

- Click the **Run** link at the top of the page.
- Click **Run** on the Run Procedure page.

Your Job Details page should be similar to the following example.



Note: The screen example above illustrates a job that is still running—notice the green circular-arrow icon at the top of the page. This icon changes to a "checkmark" if this job completes successfully.

Scenario extension - Using postsp

ElectricCommander implements data collection with a postprocessor. The postprocessor is a command associated with a particular procedure step. If the postprocessor is specified for a step, it executes concurrently with the main step command. The postprocessor runs on the same machine as the main command and in the same working directory, and it retrieves the log file from the step as its standard input.

The standard Commander postprocessor is called **postp**. **postp** scans the step's log file looking for interesting output such as error messages and then sets properties on the job step to describe what it found. For example, postp might create a property named "errors" whose value is the number of error messages in the log file, or a property named "tests" that counts the number of tests the step executes. Also, postp can extract portions of the step log that contain useful diagnostic information and save this information for reporting.

When you create a step, you can specify `postp`. If you already have a step and want to add postp reporting, you just have to edit the step to include postp.

- From the previous Job Details page screen example, click the **Edit** link for the AntBuild step.
- On the Edit Step - AntBuild page, in the Parameters section, add `postp` to Postp Line field.

The screenshot shows the 'Edit Step - AntBuild' interface. The 'General' section includes fields for 'Name' (AntBuild) and 'Description'. The 'Subprocedure' section shows 'Subprocedure: EC-Ant : runAnt' and a 'Resource' field with a 'Browse' button. The 'Parameters' section contains various configuration options: 'Ant Location', 'Build File' (build.xml), 'Libraries', 'Properties', 'Property File', 'Debug' (checkbox), 'Diagnostics' (checkbox), 'Output Level' (radio buttons for Normal, Quiet, Verbose), 'Log File', 'Target' (\$[target]), 'Additional Commands', 'Postp Line' (highlighted with a red circle and containing 'postp'), and 'Working Directory' (project).

Click **OK** to save your edit and return to the Job Details page.

Run this job again to see the postp result—click the **Run/Run Again** link.

On the next screen example, notice the AntBuild step now shows "3 compiles" in the Status column. Using postp supplies additional information.

Job Details – job_1934_201104181440
 Run Again Delete Save Configuration >>

Completed with Success
 Start Time: 2011-04-18 14:40:08 PDT
 Elapsed Time: 00:00:40.900

General Information
 Project: FirstNewProject
 Procedure: MultiAgentBuild
 Launched by: admin
 Priority: normal

Steps | Diagnostics | Parameters | Properties | Notifiers

View: All Expand All | Collapse All

Step Name	Log	Status	Elapsed Time	Resource	Actions
[-] checkout		✓ Success	00:00:13.069		Edit
[-] checkoutMethod		✓ Success	00:00:13.066		Edit
runMethod		✓ Success	00:00:13.158	local	Edit
[-] AntBuild		✓ Success	00:00:14.186		Edit
createCommandLine		✓ Success	00:00:05.284	local	Edit
runCommandLine		✓ 3 compiles	00:00:08.237	local	Edit
[-] unittest		✓ Success	00:00:06.999		Edit
createCommandLine		✓ Success	00:00:06.100	local	Edit
runCommandLine		✓ Success	00:00:00.447	local	Edit
[-] systemtest		✓ Success	00:00:05.311		Edit
createCommandLine		✓ Success	00:00:04.319	local	Edit
runCommandLine		✓ Success	00:00:00.419	local	Edit
seeWorkspace		✓ Success	00:00:00.356	local	Edit

Records per page: 100
 1 thru 13 of 13

For more information about using postp, see the ["Postprocessors"](#) Help topic.

Summary

This scenario demonstrated how ElectricCommander can drive a complete build and test process—including building steps to run in parallel and creating a resource pool. This scenario also provided a brief look at how you can use Commander to execute rapid root cause analysis by reviewing postp diagnostics.

Configuring ElectricCommander

Use this page for quick access to configuring resources, workspaces, email configurations, your source control system, and defect tracking to communicate with ElectricCommander.

Note: If you are viewing this help topic from the "Configuring ElectricCommander" web page, use these instructions:

Click the **Add** link for any object you want to configure and that objects configuration page is displayed.

- Each item you configure will be listed in this table for your easy reference.
- Configure objects listed in this table are linked to their respective "**Edit...**" page in the event you need to modify any values previously supplied. Select a configured object to edit its values.

The table below provides a brief description of Commander configuration tasks

Click on one of the "**Setting Up...**" links to see more information about that topic.

Setting Up Resources	Before resources can be set up, you need to know which agent machines are available to allocate to jobs. The Commander Resource web page displays all resources currently available to the ElectricCommander server.
Setting Up Workspaces	<p>ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a <i>job workspace</i>.</p> <p>A job step can create whatever files it needs within its workspace and Commander automatically places files, such as step logs, in the workspace. The location of the job step workspace is displayed on the Job Details web page.</p>
Setting Up Email Configurations	<p>If you do not create an Email Configuration, you will be unable to send email notifications to individuals or groups.</p> <p>If you have multiple users or groups in remote locations who use a different mail server, you will want to create additional email configurations to accommodate those locations if they need to receive notifications.</p>

Setting Up a Source Control Configuration	Commander needs to communicate with your Source Control system (SCM) if you intend to use the CI Dashboard for continuous integration schedules to start a new build based on new/modified source control checkins, or if you plan to configure Preflight builds (to build and test code changes before those changes are "committed").
Setting Up Directory Providers	Commander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. Local users and groups can be defined within Commander.
Setting Up Defect Tracking	Commander uses plugins to integrate with numerous defect tracking systems. If the plugin you need for your defect tracking system is not automatically installed with Commander, see the Plugin Catalog to find the integration you need.

Web Interface Online Help System

Use ElectricCommander's online help system for more information. Click the **Help** link in the top-right corner of any product web page to see a specific help topic for that page.

When the help system opens, Electric Cloud recommends reviewing the Help table of contents. All Help folders above the Web Interface Help folder are user-guide style help topics that provide more detailed information on each of their topics. If you generally prefer to use a command line rather than the Commander web interface, you will find complete ectool (the Commander command-line tool) commands and arguments here too.

Setting Up Resources

Before resources can be configured, you need to know which agent machines are available for allocation to jobs. The Resources page displays all resources currently available to the ElectricCommander server. This page is your Resources management center—you to create, modify, or delete resources, see resource status, pools or zones where resources reside, and much more.

Selecting the Cloud tab displays the Resources page.

What is a resource?

A *resource* specifies an agent machine where steps can execute and...

- each resource has a logical name
- each resource refers to an agent machine by its host name
- several resources can correspond to the same physical host
- each resource can be assigned to one or more pools, or one zone

A pool is a group of interchangeable resources. For example, you might have a pool of Windows servers.

Naming a pool in a procedure step allows Commander to assign that step to any resource in the pool, which means Commander can choose a lightly-loaded resource and a pool makes it easy for you to change your server configuration.

- you can specify a resource or a pool name in a procedure step to execute that step on that resource/pool
- you can define custom properties for your resources or resource pools

For example, if configuration information varies from resource to resource, you can use custom properties to hold this information and then reference it from procedure steps where it is needed. Suppose you have a pool of test machines where the test hardware is in a different location on each machine. You could define a property named "testLocation" on each resource, then pass this property to procedure steps using a reference such as `$/myResource/testLocation`. This approach allows you to define procedure steps to run on any resource and automatically handle configuration differences.

Commander supports two resource types

- Standard
- Proxy

Standard Resources

This resource type specifies a machine running an installed Commander agent on one of the supported agent platforms, as specified in the *ElectricCommander Installation Guide*, chapter 2, "System Requirements and Supported Platforms."

Proxy Resources

This resource type requires SSH keys for authentication. You can create proxy resources (agents and targets) for Commander to use on numerous other remote platforms/hosts that exist in your environment. A proxy agent

is a Commander agent, channeling to a proxy target.

- **Proxy agent** - This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform. A proxy agent is a Commander agent, channeling to a proxy target. Proxy agents have limitations, such as the inability to work with plugins or communicate with ectool commands.???
- **Proxy target** - This is a machine on an unsupported platform that can run commands via an SSH server.

When a step is running on a proxy resource, the proxy agent performs the following tasks:

- Uploads the `commandFile` to the Working Directory on the proxy target via SSH
 - Working Directory - the step runs in this directory on the proxy target, which defaults to the UNIX path to the workspace
- Creates a wrapper `sh` shell script to:
 - CD to the Working Directory
 - set `COMMANDER_environment` variables that exist in the proxy agent's environment
 - run the `commandFile` that was uploaded earlier
- Uploads the wrapper script to the Working Directory on the proxy target
- Runs the wrapper script on the proxy target
- Cleans up script files on the agent and proxy target

Setting up SSH keys

If you are not familiar with setting up SSH keys, go to the knowledge-based article called "Setting up SSH Keys in preparation to use a Proxy Agent" in the Electric Cloud Support Web Site:

1. Go to <https://electriccloud.zendesk.com/hc/en-us/categories/200176553-ElectricCommander>.
2. In the Commander KB section, enter **KBEC-00049** in the Search field.
3. Click **Search**.
4. Click the link to the knowledge-based article called "KBEC-00049 - Setting up SSH Keys in preparation to use a Proxy Agent" to open it.

On UNIX, the Commander agent looks in the `~/ .ssh` directory for the private and public key files to set up the SSH connection to the proxy target. Therefore, Electric Cloud recommends generating your key files in that directory.

On Windows, the Commander agent does not presume a default location. Therefore, in the proxy resource definition you must specify the key file location. Enter the following line in the resource "Proxy Customizations" field:

```
setSSHKeyFiles('c:\ foo\priv.key');
```

To create a new resource

Click the Cloud tab in the Commander web interface to go to the Resources page.



At the top right-side of the table, click the icon to select the **Create Resource** or **Create Proxy Resource** link to see the New Resource or New Proxy Resource panel.

From either panel, New Resource or New Proxy Resource, click the **Help** link in the upper-right corner of the page to access the [Resources](#) Help topic to see descriptions and other information to assist with filling-in the fields.

Gateways and zones

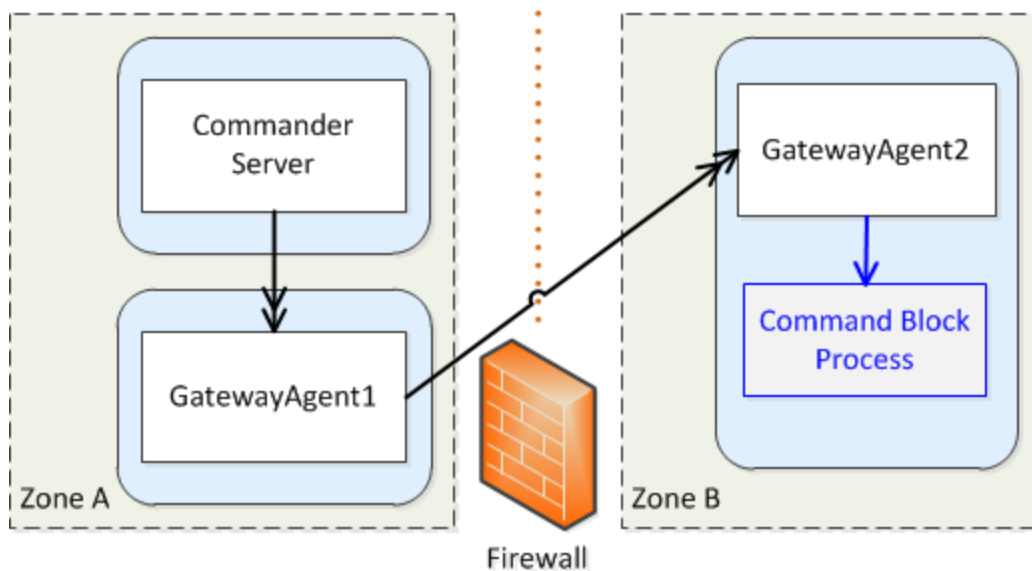
ElectricCommander provides the capability of establishing security zones for agents and repository servers.

What is a zone?

A zone consists of a collection of resources that can directly communicate with each other. Within a large corporation, a zone may encapsulate a physical site, and communication between the two sites occurs through firewalls.

Cross-zone communication

The following figure shows an example gateway and zone configuration with two zones and three machines.



Cross-zone communication is managed by proxying requests through specially marked resources called “gateway resources”. In the previous example, if the Commander server wants to run a command block process step on an agent in Zone B, it issues its request through GatewayAgent1. The request includes extra metadata that instructs GatewayAgent1 to forward the request to the target agent. The firewall is configured to allow connections from GatewayAgent1 to GatewayAgent2. It is also configured to allow connections from GatewayAgent2 to GatewayAgent1. This is necessary because API communication (CLI commands as well as the step completion message) is sent from GatewayAgent2 on its own outbound connection to the server (through GatewayAgent1).

Because the firewall separates two private networks, there's no guarantee that IP address ranges in one network do not overlap with those of the other. When either gateway agent wants to connect to the other, the gateway agent uses an IP address to the firewall that is valid for its side of the firewall.

What is a gateway?

The details of this connection information are recorded in a gateway object in Commander. Using the previous example as a reference, a gateway object consists of the following information:

- Name of a gateway resource in one zone (GatewayAgent1)
- Name of a gateway resource in another zone (GatewayAgent2)
- Host/port GatewayAgent2 uses to communicate with GatewayAgent1 (e.g. the firewall IP address in ZoneB)
- Host/port GatewayAgent1 uses to communicate with GatewayAgent2 (e.g. the firewall IP address in ZoneA)

Support for gateways

The previous example shows two zones for simplicity, but Commander supports an unlimited number of zones, including chained zones. For example, if a third zone called Zone C is only accessible from Zone B via GatewayAgent3 (in Zone B) and GatewayAgent4 (in Zone C), the server could issue a request to GatewayAgent1, which forwards the request to GatewayAgent2, which forwards it to GatewayAgent3, which forwards it to GatewayAgent4.

Also, for resiliency, Commander supports having multiple gateway agents between two zones. If one gateway agent goes down, the system will detect the failure and route all requests through the other gateway agent.

Setting Up Workspaces

ElectricCommander provides each job step with an area on the disk 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 within its workspace, and Commander automatically places files in the workspace, such as step logs. Normally, a single workspace is shared by all steps in a job, but it is possible for different steps within a job to use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

To create a new workspace

Commander can handle any number of workspaces.

- To define a workspace from the web interface, click the Cloud > Workspaces tabs.
- On the Workspaces page, click the **Create Workspace** link at the top right-side of the table and the New Workspace web page is displayed.

Click the **Help** link in the top-right corner of the web page for assistance with filling in the fields to create a workspace.

To see the Workspaces Help topic now, click [Workspace - create new or edit existing workspace](#).

After you create a workspace specification, you will see it listed on the Workspaces page.

Setting Up Email Configurations

If you do not create an Email Configuration, you cannot send Email Notifications to individuals or groups.

If you have multiple users or groups in remote locations who use a different mail server, you will want to create additional Email Configurations to accommodate those locations if they need to receive Commander notifications.

To create an email configuration

To create an Email Configuration, click on the Administration > Email Configurations tabs, then click the **Add Configuration** link to see the New Email Configuration page. Click the **Help** link in the upper-right corner of the page to see descriptions and other information to assist with filling-in the fields.

To see the Email Configurations Help topic now, click [Email Configuration - create new or edit existing email configuration](#).

After creating an Email Configuration, you will see it listed in the table on the Email Configurations web page.

Setting Up a Source Control Configuration

You must configure your source control system to communicate with the ElectricCommander server if you plan to take advantage of the Commander Continuous Integration Dashboard and associated functionality and Preflight functionality—both of these features are designed to work with a number of source control (SCM) systems.

- Continuous Integration Dashboard - use this feature to create continuous integration schedules for your build environment. The Continuous Integration Manager is the front-end user interface for the ElectricSentry Continuous Integration engine.
- Preflight - use this feature to build and test developer code before it is *committed* to the code base for your product.

Commander installs (bundles) and supports numerous source control types. After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page—to see this web page, select the Administration > Source Control tabs.

Note: If the SCM you prefer is not listed here, see the Plugin Manager page (Administration > Plugins) for a list of all currently available SCM plugin integrations available for Commander. To configure a different SCM, see the help topic associated with that plugin.

Select one of the following links to go to the source control configuration page to configure your SCM system:

[AccuRev](#)

[ClearCase](#)

[File](#)

[Git](#)

[Perforce](#)

[Property](#)

[Subversion](#)

The Continuous Integration Dashboard has its own help topic. For more information on ElectricSentry or Preflight builds, see their respective help topics: [ElectricSentry](#) , [Preflight Builds](#).

Setting Up Directory Providers

ElectricCommander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. Local users and groups can be defined within Commander.

To specify a directory provider

ElectricCommander includes a web page to facilitate adding your existing LDAP or Active Directory users and groups to Commander.

- To access the Directory Providers web page, select the Administration > Directory Providers tabs.
- Click the **Add Active Directory Provider** or the **Add LDAP Provider** link to add a new provider.

Depending on which **Add... Provider** link you choose, you will see either the New Active Directory Provider or the New LDAP Provider web page. From either page, click the **Help** link in the upper-right corner of the page. The Help topic describes each field on either web page so you can easily fill-in the forms to have Commander use your existing user or group account information.

To see the Directory Provider Help topic now, click [Directory Providers - create new or edit existing directory providers](#).

After adding a provider, you will see it listed in the table on the Directory Providers web page.

Enable/Disable Local Commander Users

Two server settings properties are supplied for local Commander users (both are "on" by default):

- `enableAdminUser` - if set to '0', attempts to log in as the 'admin' user are denied as if the user did not exist
- `enableLocalUsers` - if set to '0', attempts by a non-admin local user to log in are denied as if the user did not exist

These settings are stored in properties in the `/server/settings` property sheet and can be set from ectool by calling:

```
ectool setProperty /server/settings/enableAdminUser 0
```

Or, these settings can be set from the Commander web interface:

- Select the Administration > Server tabs.
- At the top of the page, select the **Settings** link.

Note: Disabling these properties takes effect only for new sessions. Existing sessions continue to function "as-is" until the user logs out and attempts to log in again.

wrapper.conf Properties

If you need to edit the wrapper.conf file, see <http://wrapper.tanukisoftware.com/doc/english/properties.html> for complete information about property formats.

The Home Page

[Job Configurations](#)

[Shortcuts](#)

[Jobs Quick View](#)

[Reports](#)

Overview

This page provides a convenient console for running jobs and viewing results.

- This web page is your Commander Dashboard for tracking project health.
- You can customize the page to display only the jobs that interest you and to provide one-click access to the jobs you run frequently. Each of the Home page sections can be customized.
- The Reports section allows you to see thumbnail report images that are updated each time the report is generated.
- This page provides shortcuts for quick access to pages you use most frequently in ElectricCommander or anywhere on the web.

IMPORTANT: If you are using or plan to use the EC-Homepage plugin to share your Home page sections, you will not see some links normally available to you. For example, if a particular Jobs Quick View category is shared, you **cannot** Delete or Modify that category. You can perform these functions only if the category belongs to you alone.

For information on sharing your Home page configuration, see the "[Home page configurations](#)" section in the "Customizing the Commander UI" help topic.

Job Configurations

Procedures in ElectricCommander can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem. When you create a job configuration, you supply all the information needed to run a procedure, including parameters and/or a credential. All job configurations are displayed here on the Home page. Invoke a particular configuration by clicking its name in the Job Configurations section.

Create Job Configurations three ways

- On the Home page, create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section.
 - In the Create Configuration popup menu, select the project and procedure you want to use for creating this configuration.

- From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page.
Your saved job configuration is displayed on your Home page.
- From the Edit Schedule page, click the **Save Configuration** link at the top of the page.
Your saved configuration is displayed on your Home page.

Shortcuts

Use shortcuts to save frequently visited Commander web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

Create Shortcuts two ways

- Mouse-over the "star" icon at the top of any ElectricCommander page and click "Add current page" to add that page to the shortcut list. Mouse-over the star icon again and click "Remove current page" to remove the shortcut for that page. The star icon is yellow for pages saved as a shortcut and gray for those pages not saved as a shortcut.
- Click the **Create** link in the Shortcut section and provide a name and URL to create a shortcut.

To modify or update a shortcut, click the **Edit** link adjacent to the shortcut you want to change.

Shortcuts can be accessed conveniently from any ElectricCommander web page. Mousing-over the star icon displays a list of shortcuts saved by the current user. Click on a shortcut name to view the page.

Note: The Shortcut section may contain static entries that cannot be deleted. And if you "share" your Home page, the Edit link will not be available for shared items.

Jobs Quick View

Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

The Jobs Quick View allows you to define job categories that are interesting to you.

The Home page displays the most recent jobs in each category, and you can easily click-through to get more details about any of those jobs. For example, clicking on a job name takes you to that job's Job Details page.

Create a job category

- Click the **Add Category** link in the Jobs Quick View section.
After creating a category, results are displayed on the Home page. Clicking the **Details** link displays a summary to the right of the category. In addition to job status and other diagnostic information, the summary displays running steps and failed steps (containing errors and warnings).
- Click the **Modify** link to edit the Jobs Quick View category or the **Delete** link to remove that job from the Jobs Quick View category.
- Click on the **Details** link to see job summary information—the summary remains visible, regardless of mouse location, until you click somewhere else on the page.

Note: If you "share" your Home page, the Modify and Delete links will **not** be available for shared items.

Reports

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

- Click the **Add Report** link to go to the New Reports page.
After filling-in the information on the New Reports page, you will see a thumbnail view of your report (after it runs) on your Home page.
Note: If the drop-down menu on this page is empty, click the **Help** link on the New Report page for more information.
- Click the Collapse/Expand (+/-) box to see the full thumbnail report or just the report title.
- Click the **Rename** link if you need to rename your report.
- Click the **Remove** link if you need to remove this report from your Home page.

Note: If you "share" your Home page, the Rename and Remove links will not be available for shared items.

Using ElectricCommander in Your Environment

This topic addresses some common issues you may encounter as you start using ElectricCommander and offers some tips about managing your Commander projects.

[What's in a step?](#)

[Where's the script for the step?](#)

[Environment variables](#)

[When do you use subprocedures?](#)

[How do you evolve procedures?](#)

[Porting existing scripts](#)

[Commander project version control](#)

What's in a step?

One of the first questions you may ask when you start to implement your build and test processes on Commander is how to divide processes into individual steps: "Should I use only a few steps, each of which does a lot, or a large number of fine-grain steps?" Here are some factors to help you decide how many steps to use:

- **Reporting** - If you would like a separate report of success/failure for two activities, put those activities in separate steps. If you are happy to have a single report for both activities, a single step may make sense. For example, compilation and test phases should probably be in different steps because errors in the two phases will probably be handled differently. Unit tests for product components managed by different groups may make sense in separate steps; each group can watch for errors in its step.
- **Parallelism** - If you want to use Commander to run two activities in parallel, put them in separate steps. If these activities are in the same step, Commander cannot run them in parallel.

Commander parallelism works best at a coarse grain, such as running different sets of unit tests in parallel or compiling for different platforms. Electric Cloud does not recommend trying to do fine-grain parallelism with Commander, such as compiling every individual source file in a separate step— this is likely to be complicated and brittle, resulting in an enormous number of job steps, which will make it difficult to view results. If you would like to use fine-grain parallelism for compilation, Electric Cloud recommends using our ElectricAccelerator product. In this case, you would make your compilation steps large, with as much work in each makefile as possible because ElectricAccelerator automatically subdivides the work and runs as many sub-steps as possible in parallel. The more work in a makefile, the more efficient the process becomes.

- **Resources** - If two different activities need to run on different resources, they need to be in different steps. A single step runs entirely on a single resource.

- **Conditional steps** - If you want to skip portions of your process during some jobs, but execute them during others, it probably makes sense to put those activities in separate steps and use the Commander "Run condition" mechanism to decide whether they run during each particular job. However, if you are invoking programs like *make* that already allow you to choose which actions to perform, it may make more sense to have a single large step and handle conditional behavior with those programs, rather than with Commander. For example, when running *make*, you can specify the targets to be built.
- **Setup** - You might want to have a single step at the beginning of a procedure that processes the procedure's parameters and sets up the environment for the rest of the procedure. The setup step will create a snapshot of your source code or set up a ClearCase view also. After setup, the remaining steps should be easier to write because they just use information created during the setup step.

Where's the script for a step?

Typically, each step executes a script of some type, which is processed by a command language such as *cmd* on Windows, a *shell* on UNIX, or Perl.

Two ways to handle script execution

- The first approach - Enter the script directly into the command field for the step. You can specify the language interpreter as the shell for the step.
- The second approach - Store the script in a file that is part of the source code for your project, then specify a simple command for the step that invokes the command language to process the script file.

Electric Cloud recommends using the second approach in most cases. The main advantage of this approach is that it makes your Commander procedures more robust. For example, suppose the script needs to change as your product evolves. If you have kept the script outside Commander with the source code for the product, each product version can have its own copy of the script. When you extract the source code for the product at the beginning of a job, you also extract the scripts for its steps. You can change a step's script without worrying about its impact on other versions of the product.

However, if the script for a step is stored in the step, it is more difficult to evolve the script with new product versions. You probably still need to build older product versions, so you will worry whether the new script for the step will work with older product versions. If the script does not work, perhaps you can modify the step's script to test the version being built and take different actions for each version—this process becomes increasingly more complex as the number of versions increases. Or, you can make a separate copy of the build procedure for each product version (more on this below), but this process also gets complicated as you acquire more and more procedure versions. We find it is easier to store scripts for steps with the product code, so script changes are handled in the same way as changes to the product.

Two cases where it makes sense to store the script for a step in the ElectricCommander step

- The first case is for the first step of a job.
At this point you have not extracted the source code for the product, so you do not yet have access to any scripts stored with the source code.

- The second case is for steps with one or two commands only, or steps already specified primarily by information stored with your source code.
For example, if a step is running a *make* or *ant* command, step behavior is already specified almost entirely by a makefile or a `build.xml` file for Ant. In this case, the step's script invokes a single, relatively simple command; no need to store this in a file. If you subsequently find that the script for a step is changing frequently, consider moving it out of Commander and into a file stored with your source code.

The object is to make your Commander procedures as reusable as possible, so no change is needed with every small change to your product. Even better, organize your Commander procedures so a single procedure can be used for multiple products. To do this, store product-or version-specific information with the product, not with Commander.

Environment variables

Your build and test scripts probably depend on certain environment variables having certain values. In your existing system, you probably set those values at the beginning of your script. However, in Commander you need to set those values in each step that depends on them. The easiest way to set values is to create a short command file during the first step of the job and save it in the top-level directory of the job workspace. The command file should contain a sequence of commands to set all environment variables required by the job. Then, in each subsequent step, invoke that command file at the beginning of the step to set environment variables for that step. This approach simplifies management of your environment variables: if you need to change a variable, you change only the setup step at the beginning of the job, and the value is reflected in all of the following steps automatically.

When do you use subprocedures?

In Commander, the action for a step can invoke another procedure, passing parameters. This is a powerful tool for structuring your processes. Subprocedures tend to be used in two ways: for *encapsulating reusable processes* and for *managing concurrency*.

- *encapsulating reusable processes*, is the preferred use. If you need to perform certain activities repeatedly in different places, you can use a subprocedure for them. For example, when we build and test the Commander product, we do it on multiple platforms, but the mechanism is virtually the same on all platforms. We implement the basic build and test mechanism for one platform in a subprocedure named `BuildAndTest`. In our main procedure for production builds, we invoke `BuildAndTest` multiple times, once for each of the platforms. If the mechanism changes, we only need to change it once in `BuildAndTest` to fix all six platforms.
- *managing concurrency* - suppose you have three steps A1, A2, and A3, that must run sequentially, and two other steps B1 and B2 that must also run sequentially, but the two groups can run in parallel. The way to implement this is to put A1, A2, and A3 in one subprocedure named "A", and B1 and B2 in another subprocedure named "B". Now you can create a top-level procedure with two steps: one invoking A and the other invoking B, and mark those steps for parallel execution. As a result, A and B will run in parallel but the steps inside each subprocedure will run serially.
If you placed all five steps inside the top-level procedure, there would be no way to achieve this effect.

How do you evolve procedures?

After you start using Commander for managing your build and test processes, it will not be long before you encounter the following situation:

You have a set of Commander procedures that work fine on all existing versions of the product, but you are about to change the product in a way that affects Commander procedures. For example, you might be adding a new component, or perhaps you are going to change the way the product is installed for testing, or perhaps you have restructured the product to allow more steps to run concurrently. As a result, you need to change Commander procedures for the current version. At the same time, you need older product versions to continue building as well.

In most cases, the easiest way to handle these situations is to leave the current Commander procedures alone and keep using them for your older product version. This ensures you will not "break" older versions. Make copies of the procedures that need to change, then modify the copies and use them for your new software version.

You will have one version of procedures for each significant version of the product. For example, at Electric Cloud, we incorporate the version number into the procedure names: "Master-1.0" is the version of the Master procedure used with the 1.0 release, "Master-1.1" is used for 1.1, and "Master" with no version number is used for our current development.

Another approach would be to make a copy of the entire project for each release. This method may be easier if other information in the project, such as property values, needs to evolve also.

The "copying" approach gets more and more complicated as you add more and more versions of the procedure. To minimize this complexity, try to structure your procedures so they do not have to change frequently. Some techniques you can use:

- Use parameters for things that do change frequently, such as the branch of software to build.
- Where possible, store scripts and other information [used by the procedure] as part of the source code for the project as described above, so this information is versioned automatically, along with your source code.

Porting existing scripts

You probably already have scripts you have been using to build and test software, before switching to ElectricCommander.

Over time, you will probably want to do quite a bit of restructuring to take full advantage of Commander. However, you probably do not need to do major restructuring to get started with Commander. Here are some steps you might take to "get up and running" with Commander as quickly as possible and gradually convert to make the best use of the system.

To begin, see if you can take your existing script and run it monolithically as a single step in a procedure under Commander. This will get you running and start providing some Commander benefits for resource management, error analysis, and reporting.

Next, start dividing your previous script into separate steps for Commander. Begin with the steps easiest to separate from the rest of the script, such as the commands to compile your software. One of the challenges you face is how to pass data from one step to another. In a monolithic script, you can keep the data in variables that persist throughout the job. As you divide the script into steps, you need to figure out which variables are used only in a single step and which variables pass from step to step. For variables that pass between steps, use Commander properties to store their values. In many cases you can compute shared

data values in a single step at the beginning of the job, store their values in properties on the job, then access those values read-only in later job steps.

After dividing the steps, you can do finer-grain reporting. Start using the *postp* postprocessor to scan your log files and generate statistics and diagnostics. Over time you will probably discover you have a few error and warning messages peculiar to your site, and are not captured by *postp*'s patterns. Learn how to extend *postp* with additional patterns to capture all the information that matters to you.

Now that reporting statistics are being recorded, you may wish to develop your own reports to summarize those statistics. You can easily use *ectool* to read statistics from the properties where they are stored.

Something else you can do after steps are divided: start tuning the performance of your procedures. For example, you can start marking steps to run in parallel, and you can choose resources on a step-by-step basis to finish your jobs faster and share resources more effectively.

Commander project version control

Electric Cloud recommends exporting your Commander project data at regular intervals and saving it in the same configuration management system you use for your source code (use the "export" *ectool* command to generate an XML file representing the contents of a project or procedure). This process allows you to track project changes and also allows you to "look back" at older versions if that should become desirable. Also, you should snapshot a version of Commander project data at the time of each major software release, so you can revert to the exact Commander configuration used to generate that release if necessary.

Using ectool and the Commander API

ElectricCommander features can be accessed in two ways:

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

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

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

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

- [Using ectool](#)
- [Using ec-perl](#)
- [Common global options](#)
- [The Batch API](#)
- [Installing Commander Perl modules into your Perl distribution](#)
- [Installing Perl modules into the Commander Perl distribution](#)

Using ectool

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

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

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

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

Logging in

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

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

Login example:

```
ectool --server bldglserver login "Ellen Ernst" "ee123"
```

General syntax for ectool command usage:

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

Global Arguments (optional)

See the [Common global options](#) section for more information.

Passing Lists as Arguments

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

For ectool

- **value** list - each value is specified as a separate argument on the command line

Example:

```
ectool addUsersToGroup group1 --userNames user1 user2 user3
```

- **name/value** pairs - each pair is specified as a separate argument in the form *name=value*

Example:

```
ectool runProcedure proj1 --procedureName procl --actualParameter parm1=value1 p  
arm2=value2
```

For ec-perl

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

Example:

```
$cmdr->addUsersToGroup({ groupName => group1,  
                        userName => ['user1', 'user2']});
```

- **name/value** pairs - the argument value is a reference to an array of hash references. Each hash contains a pair of entries, one for the name and one for the value. The hash keys depend on the specific API.

Example:

```
$cmdr->runProcedure({ projectName => 'proj1',  
                    procedureName => 'procl',  
                    actualParameter => [{ actualParameterName => 'parm1',  
                                         value => 'value1'},  
                                         { actualParameterName => 'parm2',  
                                         value => 'value2'}]});
```

Using Perl

When ElectricCommander is installed—Server, Agent, or Tools (using the express or advanced installation type)—a copy of Perl is installed. This Perl is pre-configured with all the packages you need to run the

Commander Perl API. Commander does not, however, automatically add this version of Perl to your path because:

- We did not want the ElectricCommander 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 ElectricCommander, and it is in a directory that is added to your path. When the `ec-perl` wrapper runs, it sets up the environment, finds, and calls the Commander copy of Perl, passing all of its parameters to Perl.

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

```
ec-perl yourPerlOptions yourPerlScript.pl
```

The Perl script can include API calls to ElectricCommander with no other special handling required.

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

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

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

Perl API structure

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

Specifying global options

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

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

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

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


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

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

To see a complete list of global commands you can use with Perl, click [here](#).

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

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

Specifying subcommands

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

For example, to retrieve a list of jobs, use

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

Specifying arguments

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

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

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

or

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

Handling return values

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

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

Error handling

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

For example:

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

An alternative to using the `abortOnError` flag:

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

Specifying a named object

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

```
$cmdr->getProject ('$[/server/defaultProject]')
```

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

Common Global Options

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

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

Global Arguments	Description
<code>--server <hostname></code>	<p>ElectricCommander server address. Defaults to the <code>COMMANDER_SERVER</code> environment variable. If this variable does not exist, the default is to the last server contacted through the API. However, if there is no record for which server was contacted, the default is to <code>localhost</code>.</p> <p>Note: If you are using multiple servers, Electric Cloud recommends using the <code>server</code> option to ensure the correct server is specified for your task. For example, if you are using the <code>import</code> API, the <code>server</code> option may be particularly important.</p> <p>Do not use in a step context: Electric Cloud recommends that steps running <code>ectool</code> or Perl scripts should never provide the <code>server</code> option if the intention is to communicate with the server that launched the step. If the intention is to communicate with a different server, this agent must be a registered, enabled resource in the second server. Thus, that server will ping the agent, and the agent will learn how to communicate with that server.</p> <p>In a step context, <code>ectool</code> and the Perl API proxy server requests through the step's agent. If the agent does not recognize the provided server-name, it rejects the request. <code>ectool</code> / Perl API retry the operation because at some point the server should ping the agent, and then the agent will have learned how to communicate with the server.</p> <p>Generally, the issue is that the server publicizes its name as a fully-qualified domain name and <code>ectool</code> / Perl API issue requests with a simple-name for the server. This can happen if the step explicitly states which server it is connecting to. Fix your steps that invoke <code>ectool</code> so they no longer include the server-name, and <code>ectool</code> will default to the server-name that the server provided.</p>
<code>--port <port></code>	HTTP listener port on the ElectricCommander server. Defaults to port 8000.
<code>--securePort <secureport></code>	HTTPS listener port on the ElectricCommander server. Defaults to port 8443.
<code>--secure <0 1></code>	<p>Use HTTPS to communicate with the Commander server.</p> <p>Note: Certain requests (for example, <code>login</code>, <code>createUser</code>, and <code>modifyUser</code>) automatically use HTTPS because passwords are being sent, which means it is not necessary to specify <code>secure</code> for those APIs. Defaults to 1.</p>

Global Arguments	Description
<code>--timeout <s></code>	An API call waits for a response from the server for a specified amount of time. Timeout for server communication defaults to 180 seconds (3 minutes) if no other time is specified. After the timeout, the API call stops waiting for a response, but the server continues to process the command.
<code>--retryTimeout <s></code>	This is a separate timer, independent of the retry flag, and used to control Commander's automatic error recovery. When the API is unable to contact the Commander server, it will keep trying to contact the server for this length of time. When the API is called from inside a step, it defaults to 24 hours.
<code>--retry <0 1></code>	Retry the request if it times out based on the "timeout" value. Default is "0" and should rarely be changed.
<code>--user <username></code>	Use the session associated with the user. Defaults to the user who last logged in.
<code>--service <spn></code>	Specify the service principal name to use for Kerberos. Defaults to HTTP@host.domain.
<code>--setDefault <0 1></code>	Use the current session as the default for subsequent invocations. Defaults to 1.
<code>encoding <charEncoding></code>	Use the specified encoding for input/output. For example, for <code>charEncoding</code> , supply UTF-8, cp 437, and so on. Default is autodetected.
<code>--dryrun</code>	Displays session information and the request that would be sent, without communicating with the server. If a subcommand is specified, the server request that would be sent is displayed. This option can also be used to change the default user/server value by specifying the <code>--user</code> or <code>--server</code> options.
<code>--silent</code>	Suppresses printing the result. For example: <code>ectool --silent createResource foo</code> will not print the resource name, agent state, any modify information, create time, owner, port, or any other information otherwise displayed when you create a resource.

Global Arguments	Description
<code>--valueOf</code>	This option can return the value of a unique element. Because many ectool APIs return an XML result, it is inconvenient to use ectool in shell scripts and makefiles where you might want a piece of the ectool result to incorporate into some other logic. Using the <code>--valueOf <path></code> option evaluates the XML result and emits the value of that node to satisfy such use cases. For example: <pre>\$ ectool --valueOf '//version' getServerStatus</pre> returns only "4.1.0.48418".
<code>--format <format></code>	Specifies the response format. Must be one of 'xml' or 'json'. Defaults to 'xml'. For example, you might specify: <pre>ectool --format json setProperty summary hello</pre>
<code>--ignoreEnvironment</code>	Force ectool to ignore <code>COMMANDER_ENV</code> variables.

The Batch API

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

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

Using the Batch API

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

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

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

Example - creating a batch object:

```
use ElectricCommander;
my $cmdr = ElectricCommander;
```

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

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

Example - creating multiple requests in a batch:

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

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

Example - submitting the batch:

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

Sample response from this example:

```
<responses xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:
    version="2.1" dispatchId=1680
    <response requestId="1">
      <property>
        <propertyId>199827</propertyId>
        <propertyName>p1</propertyName>
        <createTime>2010-07-21T16:41:20.003Z</createTime>
        <expandable>1</expandable>
        <lastModifiedBy>project: EA Articles</lastModifiedBy>
        <modifyTime>2010-07-21T16:41:20.003Z</modifyTime>
        <owner>project: EA Articles</owner>
        <value>99</value>
      </property>
    </response>
    <response requestId="2">
      <property>
        <propertyId>199828</propertyId>
        <propertyName>p2</propertyName>
        <createTime>2010-07-21T16:41:20.019Z</createTime>
        <expandable>1</expandable>
        <lastModifiedBy>project: EA Articles</lastModifiedBy>
        <modifyTime>2010-07-21T16:41:20.019Z</modifyTime>
        <owner>project: EA Articles</owner>
        <value>1</value>
      </property>
    </response>
  </responses>
```

To extract information from the response to a request, use standard XPath syntax, and supply the `requestId` returned by that specific API call to either the `find` or `findvalue` functions on the batch object.

Example - extracting response information:

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

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

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

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

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

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

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

Installing Commander Perl modules into Your Perl Distribution

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

```
tar xzvf ElectricCommander-<your version>.tar.gz
cd ElectricCommander-<your version>
perl Makefile.PL
make install;# Use nmake on Windows
```

These commands install the Commander Perl and all of its submodules. If some prerequisite modules are missing, the `Makefile.PL` script will indicate which modules are needed.

Installing Perl Modules into the Commander Perl Distribution

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

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

For Linux

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

For Windows

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

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

Use Windows Visual Studio VC6 (the same version used by Commander).

Make sure that `cl` and `nmake` are both in your path. The Visual Studio install has a Command Prompt with these executables already in the path.

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

- Use a network drive to eliminate references to spaces.

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

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

Start CPAN from the new location:

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

Configure CPAN to install into the new location:

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

Install the module:

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

Ending CPAN:

```
cpan> quit
```

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

For example:

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

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

When Upgrading Commander

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

API Commands - listed by group and in alphabetical order

Use the help system's Search feature to quickly locate the API commands you need, or click one of the links below to view the entire command list "by group" or all commands in alphabetical order. In both lists, each API command is linked to its own table "section" containing descriptions, definitions, and examples, specific to that API command.

[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.

[API commands in alphabetical order](#)

This section is one continuous list of API commands in alphabetical order.

API command tables

These sections contains expanded information for each API command, including descriptions, optional arguments, and usage examples.

The tables in this section are alphabetized by API command within each group.

Note: The API tables display positional arguments for each command; however, you can use "value pairs" to construct your command scripts if you prefer.

For more information, see the "[Using ectool and the Commander API](#)" help topic.

API commands listed by group

Click on a command to go to expanded information for that command, including its associated arguments (required and other options), descriptions, usage examples, and related commands.

ACL Management (access control list)

Commands	Description
breakAclInheritance	Breaks ACL (access control list) inheritance at the given object.
checkAccess	Checks ACL (access control list) permission information associated with an object (including inherited ACLs) for the current user.

Commands	Description
<code>createAclEntry</code>	Creates an ACE (access control list entry) on an object for a given principal.
<code>deleteAclEntry</code>	Deletes an ACE on an object for a given principal.
<code>getAccess</code>	Retrieves ACL information associated with an object, including inherited ACLs.
<code>getAclEntry</code>	Retrieves an ACE on an object for a given principal.
<code>modifyAclEntry</code>	Modifies an ACE on an object for a given principal.
<code>restoreAclInheritance</code>	Restores ACL inheritance at the given object.

Artifact Management

Commands	Description
<code>addDependentsToArtifactVersion</code>	Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.
<code>cleanupArtifactCache</code>	Deletes stale artifact versions from an artifact cache. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.
<code>cleanupRepository</code>	Deletes stale artifact versions from the repository backing-store. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.
<code>createArtifact</code>	Creates a new artifact.
<code>createRepository</code>	Creates a repository for one or more artifacts.
<code>deleteArtifact</code>	Deletes an existing artifact element and all artifact versions.
<code>deleteArtifactVersion</code>	Deletes artifact version metadata from the Commander database. (This API call does not delete or remove artifacts stored on the repository machine.)

Commands	Description
<code>deleteRepository</code>	Deletes artifact repository metadata from the Commander database. (This API call does not delete or remove artifacts stored on the repository machine.)
<code>findArtifactVersions</code>	This command returns the most current artifact version that matches the filter criteria and its dependent artifact versions. This API implicitly searches for artifact versions in the "available" state, and if run in a job step, registers the step as a retriever for the returned artifact versions. Use the Perl API for the <code>findArtifactVersions</code> command.
<code>getArtifact</code>	Retrieves an artifact by its name.
<code>getArtifacts</code>	Retrieves all artifacts in the system.
<code>getArtifactVersion</code>	Retrieves an artifact version by its name.
<code>getArtifactVersions</code>	Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.
<code>getManifest</code>	Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.
<code>getRepositories</code>	Retrieves all artifact repository objects known to the Commander server.
<code>getRepository</code>	Retrieves an artifact repository by its name.
<code>modifyArtifact</code>	Modifies an existing artifact.
<code>modifyArtifactVersion</code>	Modifies an existing artifact version.
<code>modifyRepository</code>	Modifies an existing artifact repository.
<code>moveRepository</code>	Moves an artifact repository in front of another, specified repository or to the end of the list.
<code>publishArtifactVersion</code>	Publishes an artifact version to an artifact repository.
<code>removeDependentsFromArtifactVersion</code>	Removes a list of dependent artifact versions from an existing artifact version.

Commands	Description
<code>retrieveArtifactVersions</code>	Retrieves the most recent artifact version, including its dependents, from an artifact repository.

Credential Management

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

Database Configuration

Commands	Description
<code>getDatabaseConfiguration</code>	Retrieves the current database configuration.
<code>setDatabaseConfiguration</code>	Sets the database configuration on the server. If the server is in bootstrap mode, these changes take effect immediately and the server attempts to start. If the server is running, these changes have no effect until the server is restarted.

Directory Provider Management

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

Email Configuration and Management

Commands	Description
<code>createEmailConfig</code>	Creates a new email configuration.
<code>deleteEmailConfig</code>	Deletes an email configuration.
<code>getEmailConfig</code>	Retrieves an email configuration by name.
<code>getEmailConfigs</code>	Retrieves all email configurations.
<code>modifyEmailConfig</code>	Modifies an existing email configuration.

Email Notifiers Management

Commands	Description
<code>createEmailNotifier</code>	Creates an email notifier on an object specified by an <code>emailNotifierSelector</code> .
<code>deleteEmailNotifier</code>	Deletes an email notifier from a property sheet container.

Commands	Description
<code>getEmailNotifier</code>	Retrieves an email notifier from a property sheet container.
<code>getEmailNotifiers</code>	Retrieves all email notifiers defined for the specified property sheet container.
<code>modifyEmailNotifier</code>	Modifies an email notifier in a property sheet container specified by an <code>emailNotifierSelector</code> .
<code>sendEmail</code>	Facilitates sending an email from the command-line or a Command Step without setting up an Email Notifier. This API is more dynamic than an email notifier because you do not need to setup some kind of a template beforehand. This API also makes sending email attachments easier than using a notifier template.

Gateway/Zone Management

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

Job Management

Commands	Description
abortAllJobs	Aborts all running jobs.
abortJob	Aborts a running job.
abortJobStep	Aborts any type of step—command step or subprocedure step.
completeJob	Completes an externally managed job.
completeJobStep	Completes an externally managed job step.
createJob	Creates an externally managed job.
createJobStep	Creates a job step in an existing job.
deleteJob	Deletes a job from the ElectricCommander database.
findJobSteps	Returns a list of job steps from a single job or from a single subprocedure job step. This API is used by the Job Details web page in the Commander UI.
getJobDetails	Retrieves complete information about a job, including details from each job step.
getJobInfo	Retrieves all information about a job, except job step information.
getJobNotes	Retrieves the notes property sheet from a job.
getJobs	Retrieves summary information for a list of jobs.
getJobsForSchedule	Retrieves jobs started by a specific schedule.
getJobStatus	Retrieves the status of a job.
getJobStepDetails	Retrieves details for a job step.
getJobStepStatus	Retrieves the status of a job step.
modifyJob	Modifies the status of an externally managed job.
modifyJobStep	Modifies the status of an externally managed job step.
moveJobs	Moves jobs from one project to another.

Commands	Description
<code>runProcedure</code>	Creates and starts a new job using a procedure directly or specified indirectly through a schedule.
<code>setJobName</code>	Sets the name of a running job.

Parameter Management

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

Plugin Management

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

Procedure Management

Commands	Description
<code>createProcedure</code>	Creates a new procedure for an existing project.
<code>createStep</code>	Creates a new procedure step.
<code>deleteProcedure</code>	Deletes a procedure, including all steps.
<code>deleteStep</code>	Deletes a step from a procedure.
<code>getProcedure</code>	Finds a procedure by its name.
<code>getProcedures</code>	Retrieves all procedures in a project.

Commands	Description
<code>getStep</code>	Retrieves a step from a procedure.
<code>getSteps</code>	Retrieves all steps in a procedure.
<code>modifyProcedure</code>	Modifies an existing procedure.
<code>modifyStep</code>	Modifies an existing step.
<code>moveStep</code>	Moves a step within a procedure.

Project Management

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

Property Management

Commands	Description
<code>createProperty</code>	Creates a regular string or nested property sheet using a combination of property path and context.
<code>deleteProperty</code>	Deletes a property from a property sheet.

Commands	Description
<code>evalScript</code>	Evaluates a script in a given context. This API is similar to <code>expandString</code> except that it evaluates the <code>value</code> argument as a Javascript block, without performing any property substitution on either the script or the result. The string value of the final expression in the script is returned as the <code>value</code> element of the response.
<code>expandString</code>	Expands property references in a string, in the current context.
<code>getProperties</code>	Retrieves all properties associated with an object.
<code>getProperty</code>	Retrieves the specified property value.
<code>incrementProperty</code>	Atomically increments the specified property value by the <code>incrementBy</code> amount. If the property does not exist, it will be created with an initial value of the <code>incrementBy</code> amount.
<code>modifyProperty</code>	Modifies a regular string or nested property sheet using a combination of property path and context.
<code>setProperty</code>	Sets the value for the specified property.

Resource Management

Commands	Description
<code>addResourcesToPool</code>	Adds resources to a specified resource pool.
<code>createResource</code>	Creates a new resource.
<code>createResourcePool</code>	Creates a pool container for resource.
<code>deleteResource</code>	Deletes a resource.
<code>deleteResourcePool</code>	Deletes a resource pool.
<code>getResource</code>	Retrieves a resource by its name.
<code>getResources</code>	Retrieves all resources.

Commands	Description
<code>getResourcesInPool</code>	Retrieves a list of resources in a pool.
<code>getResourcePool</code>	Retrieves a specified resource pool by name.
<code>getResourcePools</code>	Retrieves a list of resource pools.
<code>getResourceUsage</code>	Retrieves resource usage information.
<code>modifyResource</code>	Modifies an existing resource.
<code>modifyResourcePool</code>	Modifies an existing resource pool.
<code>pingAllResources</code>	Pings all resources.
<code>pingResource</code>	Pings one resources.
<code>removeResourcesFromPool</code>	Removes resources from a specified resource pool.

Schedule Management

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

Server Management

Commands	Description
<code>getVersions</code>	Retrieves server version information.

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

User/Group Management

Commands	Description
<code>login</code>	Logs into the server and saves the session ID for subsequent ectool use. The user name provided determines the permissions for commands that can be run during the session.
<code>logout</code>	Logs out of the client session.
<code>addUsersToGroup</code>	Adds ones or more specified users to a particular group.
<code>createGroup</code>	Creates a new local group of users.
<code>createUser</code>	Creates a new local user.
<code>deleteGroup</code>	Deletes a local group.
<code>deleteUser</code>	Deletes a local user.
<code>getGroup</code>	Retrieves a group by its name.

Commands	Description
<code>getGroups</code>	Retrieves all groups.
<code>getUser</code>	Retrieves a user by name.
<code>getUsers</code>	Retrieves all users.
<code>modifyGroup</code>	Modifies an existing group.
<code>modifyUser</code>	Modifies an existing user.
<code>removeUsersFromGroup</code>	Removes one or more users from a particular group.

Workflow Definition Management

Commands	Description
<code>createStateDefinition</code>	Creates a new state definition for a workflow definition.
<code>createTransitionDefinition</code>	Creates a new transition definition for a workflow definition.
<code>createWorkflowDefinition</code>	Creates a new workflow definition for a project.
<code>deleteStateDefinition</code>	Deletes a state definition.
<code>deleteTransitionDefinition</code>	Deletes a transition definition.
<code>deleteWorkflowDefinition</code>	Deletes a workflow definition, including all state and transition definitions.
<code>getStateDefinition</code>	Finds a state definition by name.
<code>getStateDefinitions</code>	Retrieves all state definitions in a workflow definition.
<code>getTransitionDefinition</code>	Finds a transition definition by name.
<code>getTransitionDefinitions</code>	Retrieves all transition definitions in a workflow definition.
<code>getWorkflowDefinition</code>	Finds a workflow definition by name.
<code>getWorkflowDefinitions</code>	Retrieves all workflow definitions in a project.

Commands	Description
<code>modifyStateDefinition</code>	Modifies an existing state definition.
<code>modifyTransitionDefinition</code>	Modifies an existing transition definition.
<code>modifyWorkflowDefinition</code>	Modifies an existing workflow definition.
<code>moveStateDefinition</code>	Moves a state definition within a workflow definition.
<code>moveTransitionDefinition</code>	Moves a transition definition within a workflow definition.

Workflow Management

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

Workspace Management

Commands	Description
<code>createWorkspace</code>	Creates a new workspace.
<code>deleteWorkspace</code>	Deletes a workspace.
<code>getWorkspace</code>	Retrieves a workspace by name.
<code>getWorkspaces</code>	Retrieves all workspaces.
<code>modifyWorkspace</code>	Modifies an existing workspace.

Miscellaneous

Commands	Description
<code>changeOwner</code>	Changes the owner of an object.
<code>clone</code>	Makes a copy of an existing ElectricCommander project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.
<code>countObjects</code>	Returns the count of objects specified by the provided filter.
<code>deleteObjects</code>	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.
<code>export</code>	Exports part or all server data to an XML file. The default is to export all data in the system—the specified path is interpreted by the server. If the path is local, it will be created on the server machine. If it is a network path, it must be accessible by the server and the server user. If it is a relative path (NOT RECOMMENDED), it must be relative to the server's working directory.
<code>findObjects</code>	Finds several different types of Commander objects—it is the underlying mechanism used to implement the Commander "Search" feature. Because of this command's general nature and the complexity of specifying filter and sort criteria, it is not supported by ectool. Use the Perl API for the <code>findObjects</code> command.

Commands	Description
getObjects	Used to retrieve the full object based on IDs returned by findObjects . All requested objects must be of the same <code>objectType</code> . See findObjects for a list of object types.
import	Imports data from an XML export file.

API commands listed in alphabetical order

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

Commands	Description
abortAllJobs	Aborts all running jobs.
abortJob	Aborts a running job.
abortJobStep	Aborts any type of step—command step or subprocedure step.
addDependentsToArtifactVersion	Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.
addResourcesToPool	Adds resources to a specific resource pool.
addUsersToGroup	Adds ones or more specified users to a particular group.
attachCredential	Attaches a credential to an object.
attachParameter	Attaches a formal parameter to a step.
breakAclInheritance	Breaks ACL (access control list) inheritance at the given object.
changeOwner	Changes the owner of an object.
checkAccess	Checks ACL (access control list) permission information associated with an object (including inherited ACLs) for the current user.

Commands	Description
<code>cleanupArtifactCache</code>	Deletes stale artifact versions from an artifact cache. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.
<code>cleanupRepository</code>	Deletes stale artifact versions from the repository backing-store. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.
<code>clone</code>	Makes a copy of an existing ElectricCommander project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.
<code>completeJob</code>	Completes an externally managed job.
<code>completeJobStep</code>	Completes an externally managed job step.
<code>completeWorkflow</code>	Marks a workflow as complete, which means transitions are no longer evaluated.
<code>createAclEntry</code>	Creates an ACE (access control list entry) on an object for a given principal.
<code>createActualParameter</code>	Creates a new actual parameter for a step that calls a nested procedure. The parameter is passed to the nested procedure when the step runs. At run time, the actual parameter name needs to match the name of a formal parameter in the nested procedure.
<code>createArtifact</code>	Creates a new artifact.
<code>createCredential</code>	Creates a new credential for a project.
<code>createDirectoryProvider</code>	Creates a new LDAP directory provider.
<code>createEmailConfig</code>	Creates a new email configuration.
<code>createEmailNotifier</code>	Creates an email notifier on an object specified by an <code>emailNotifierSelector</code> .
<code>createFormalParameter</code>	Creates a new formal parameter for a procedure.
<code>createGateway</code>	Creates a new gateway.

Commands	Description
<code>createGroup</code>	Creates a new local group of users.
<code>createJob</code>	Creates an externally managed job.
<code>createJobStep</code>	Creates a job step in an existing job.
<code>createProcedure</code>	Creates a new procedure for an existing project.
<code>createProject</code>	Creates a new project.
<code>createProperty</code>	Creates a regular string or nested property sheet using a combination of property path and context.
<code>createRepository</code>	Creates a repository for one or more artifacts.
<code>createResource</code>	Creates a new resource.
<code>createResourcePool</code>	Creates a pool container for resources.
<code>createSchedule</code>	Creates a new schedule.
<code>createStateDefinition</code>	Creates a new state definition for a workflow definition.
<code>createStep</code>	Creates a new procedure step.
<code>createTransitionDefinition</code>	Creates a new transition definition for a workflow definition.
<code>createUser</code>	Creates a new local user.
<code>createWorkflowDefinition</code>	Creates a new workflow definition for a project.
<code>createWorkspace</code>	Creates a new workspace.
<code>createZone</code>	Creates a new zone.
<code>deleteAclEntry</code>	Deletes an ACE on an object for a given principal.
<code>deleteActualParameter</code>	Deletes an actual parameter.
<code>deleteArtifact</code>	Deletes an existing artifact element and all artifact versions.

Commands	Description
<code>deleteArtifactVersion</code>	Deletes artifact version metadata from the Commander database. (This API call does not delete or remove artifacts stored on the repository machine.)
<code>deleteCredential</code>	Deletes a credential.
<code>deleteDirectoryProvider</code>	Deletes an LDAP directory provider.
<code>deleteEmailConfig</code>	Deletes an email configuration.
<code>deleteEmailNotifier</code>	Deletes an email notifier from a property sheet container.
<code>deleteFormalParameter</code>	Deletes a formal parameter.
<code>deleteGateway</code>	Deletes a gateway.
<code>deleteGroup</code>	Deletes a local group.
<code>deleteJob</code>	Deletes a job from the ElectricCommander database.
<code>deleteLicense</code>	Deletes a license.
<code>deleteObjects</code>	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.
<code>deletePlugin</code>	Deletes an existing plugin object without deleting the associated project or files.
<code>deleteProcedure</code>	Deletes a procedure, including all steps.
<code>deleteProject</code>	Deletes a project, including all procedures, procedure steps, and jobs.
<code>deleteProperty</code>	Deletes a property from a property sheet.
<code>deleteRepository</code>	Deletes artifact repository metadata from the Commander database. (This API call does not delete or remove artifacts stored on the repository machine.)
<code>deleteResource</code>	Deletes a resource.

Commands	Description
<code>deleteResourcePool</code>	Deletes a resource pool.
<code>deleteSchedule</code>	Deletes a schedule.
<code>deleteStateDefinition</code>	Deletes a state definition.
<code>deleteStep</code>	Deletes a step from a procedure.
<code>deleteTransitionDefinition</code>	Deletes a transition definition.
<code>deleteUser</code>	Deletes a local user.
<code>deleteWorkflow</code>	Deletes a workflow, including all states and transitions.
<code>deleteWorkflowDefinition</code>	Deletes a workflow definition, including all state and transition definitions.
<code>deleteWorkspace</code>	Deletes a workspace.
<code>deleteZone</code>	Deletes a zone.
<code>detachCredential</code>	Detaches a credential from an object.
<code>detachParameter</code>	Detaches a formal parameter from a step.
<code>evalScript</code>	<p>Evaluates a script in a given context. This API is similar to <code>expandString</code> except that it evaluates the <code>value</code> argument as a Javascript block, without performing any property substitution on either the script or the result.</p> <p>The string value of the final expression in the script is returned as the <code>value</code> element of the response.</p>
<code>expandString</code>	Expands property references in a string, in the current context.
<code>export</code>	Exports part or all server data to an XML file. The default is to export all data in the system—the specified path is interpreted by the server.

Commands	Description
<code>findArtifactVersions</code>	This command returns the most current artifact version that matches the filter criteria and its dependent artifact versions. This API implicitly searches for artifact versions in the "available" state, and if run in a jobstep, registers the step as a retriever for the returned artifact versions. Use the Perl API for the <code>findArtifactVersions</code> command.
<code>findJobSteps</code>	Returns a list of job steps from a single job or from a single subprocedure job step. This API is used by the Job Details web page in the Commander UI.
<code>findObjects</code>	This command finds several different types of Commander objects—it is the underlying mechanism used to implement the Commander "Search" feature. Because of this command's general nature and the complexity of specifying filter and sort criteria, it is <i>not supported</i> by ectool. Use the Perl API for the <code>findObjects</code> command.
<code>getAccess</code>	Retrieves ACL information associated with an object, including inherited ACLs.
<code>getAclEntry</code>	Retrieves an ACE on an object for a given principal.
<code>getActualParameter</code>	Retrieves an actual parameter by its name.
<code>getActualParameters</code>	Retrieves all actual parameters from a job, jobstep, or step.
<code>getAdminLicense</code>	Retrieves the admin license, which can be used when all concurrent user licenses are in use.
<code>getArtifact</code>	Retrieves an artifact by its name.
<code>getArtifacts</code>	Retrieves all artifacts in the system.
<code>getArtifactVersion</code>	Retrieves an artifact version by its name.
<code>getArtifactVersions</code>	Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.
<code>getCredential</code>	Finds a credential by name.
<code>getCredentials</code>	Retrieves all credentials in a project.

Commands	Description
getDatabaseConfiguration	Retrieves the current database configuration.
getDirectoryProvider	Retrieves an LDAP directory provider by name.
getDirectoryProviders	Retrieves all LDAP directory providers.
getEmailConfig	Retrieves an email configuration by name.
getEmailConfigs	Retrieves all email configurations.
getEmailNotifier	Retrieves an email notifier from a property sheet container.
getEmailNotifiers	Retrieves all email notifiers defined for the specified property sheet container.
getFormalParameter	Retrieves a formal parameter by its name.
getFormalParameters	Retrieves all formal parameters from a procedure, schedule, or step.
getFullCredential	Finds a credential by name, including password, from within a running step.
getGateway	Finds a gateway by name.
getGateways	Retrieves all gateways.
getGroup	Retrieves a group by its name.
getGroups	Retrieves all groups.
getJobDetails	Retrieves complete information about a job, including details from each job step.
getJobInfo	Retrieves all information about a job, except job step information.
getJobNotes	Retrieves the notes property sheet from a job.
getJobs	Retrieves summary information for a list of jobs.
getJobsForSchedule	Retrieves jobs started by a specific schedule.
getJobStatus	Retrieves the status of a job.

Commands	Description
<code>getJobStepDetails</code>	Retrieves details for a job step.
<code>getJobStepStatus</code>	Retrieves the status of a job step.
<code>getLicense</code>	Retrieves information for one license.
<code>getLicenses</code>	Retrieves all license data.
<code>getLicenseUsage</code>	Retrieves the current license usage.
<code>getManifest</code>	Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.
<code>getObjects</code>	The <code>getObjects</code> command is used to retrieve the full object based on IDs returned by <code>findObjects</code> . All requested objects must be of the same <code>objectType</code> . See <code>findObjects</code> for a list of object types.
<code>getPlugin</code>	Retrieves an installed plugin.
<code>getPlugins</code>	Retrieves all installed plugins.
<code>getProcedure</code>	Finds a procedure by its name.
<code>getProcedures</code>	Retrieves all procedures in a project.
<code>getProject</code>	Finds a project by its name.
<code>getProjects</code>	Retrieves all projects.
<code>getProperties</code>	Retrieves all properties associated with an object.
<code>getProperty</code>	Retrieves the specified property value.
<code>getRepositories</code>	Retrieves all artifact repository objects known to the Commander server.
<code>getRepository</code>	Retrieves an artifact repository by its name.
<code>getResource</code>	Retrieves a resource by its name.
<code>getResources</code>	Retrieves all resources.

Commands	Description
getResourcesInPool	Retrieves a list of resources in a pool.
getResourcePool	Retrieves a specified resource pool by name.
getResourcePools	Retrieves a list of resource pools.
getResourceUsage	Retrieves resource usage information.
getSchedule	Retrieves a schedule by its name.
getSchedules	Retrieves all schedules.
getServerStatus	Returns the status of the server.
getState	Finds a state by name.
getStates	Retrieves all states in a workflow.
getStateDefinition	Finds a state definition by name.
getStateDefinitions	Retrieves all state definitions in a workflow definition.
getStep	Retrieves a step from a procedure.
getSteps	Retrieves all steps in a procedure.
getTransition	Finds a transition by name.
getTransitions	Retrieves all transitions in a workflow.
getTransitionDefinition	Finds a transition by name.
getTransitionDefinitions	Retrieves all transition definitions in a workflow definition.
getUser	Retrieves a user by name.
getUsers	Retrieves all users.
getVersions	Retrieves server version information.
getWorkflow	Finds a workflow by name.
getWorkflows	Retrieves all workflow instances in a project.

Commands	Description
<code>getWorkflowDefinition</code>	Finds a workflow definition by name.
<code>getWorkflowDefinitions</code>	Retrieves all workflow definitions in a project.
<code>getWorkspace</code>	Retrieves a workspace by name.
<code>getWorkspaces</code>	Retrieves all workspaces.
<code>getZone</code>	Finds a zone by name.
<code>getZones</code>	Retrieves all zones.
<code>import</code>	Imports data from an XML export file.
<code>importLicenseData</code>	Imports one or more licenses.
<code>incrementProperty</code>	Atomically increments the specified property value by the <code>incrementBy</code> amount. If the property does not exist, it will be created with an initial value of the <code>incrementBy</code> amount.
<code>installPlugin</code>	Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.
<code>login</code>	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.
<code>logout</code>	Logs out of the client session.
<code>modifyAclEntry</code>	Modifies an ACE on an object for a given principal.
<code>modifyActualParameter</code>	Modifies an existing actual parameter. An actual parameter is a name/value pair that is passed to a subprocedure. This command supports renaming the actual parameter and setting its value.
<code>modifyArtifact</code>	Modifies an existing artifact.
<code>modifyArtifactVersion</code>	Modifies an existing artifact version.
<code>modifyCredential</code>	Modifies an existing credential.
<code>modifyDirectoryProvider</code>	Modifies an existing LDAP directory provider.

Commands	Description
<code>modifyEmailConfig</code>	Modifies an existing email configuration.
<code>modifyEmailNotifier</code>	Modifies an email notifier in a property sheet container specified by an <code>emailNotifierSelector</code> .
<code>modifyFormalParameter</code>	Modifies an existing formal parameter.
<code>modifyGateway</code>	Modifies an existing gateway.
<code>modifyGroup</code>	Modifies an existing group.
<code>modifyJob</code>	Modifies the status of an externally managed job.
<code>modifyJobStep</code>	Modifies the status of an externally managed job step.
<code>modifyPlugin</code>	Modifies an existing plugin.
<code>modifyProcedure</code>	Modifies an existing procedure.
<code>modifyProject</code>	Modifies an existing project.
<code>modifyProperty</code>	Modifies a regular string or nested property sheet using a combination of property path and context.
<code>modifyRepository</code>	Modifies an existing artifact repository.
<code>modifyResource</code>	Modifies an existing resource.
<code>modifyResourcePool</code>	Modifies an existing resource pool.
<code>modifySchedule</code>	Modifies an existing schedule.
<code>modifyStateDefinition</code>	Modifies an existing state definition.
<code>modifyStep</code>	Modifies an existing step.
<code>modifyTransitionDefinition</code>	Modifies an existing transition definition.
<code>modifyUser</code>	Modifies an existing user.
<code>modifyWorkflowDefinition</code>	Modifies an existing workflow definition.
<code>modifyWorkspace</code>	Modifies an existing workspace.

Commands	Description
<code>modifyZone</code>	Modifies an existing zone.
<code>moveDirectoryProvider</code>	Moves an LDAP directory provider in front of another specified provider or to the end of the list.
<code>moveJobs</code>	Moves jobs from one project to another project.
<code>moveRepository</code>	Moves an artifact repository in front of another, specified repository or to the end of the list.
<code>moveStateDefinition</code>	Moves a state definition within a workflow definition.
<code>moveStep</code>	Moves a step within a procedure.
<code>moveTransitionDefinition</code>	Moves a transition definition within a workflow definition.
<code>pingAllResources</code>	Pings all resources.
<code>pingResource</code>	Pings one resources.
<code>promotePlugin</code>	Sets the promoted flag on a plugin. Only one version of a plugin can be promoted at a time, so setting the promoted flag to <code>true</code> on one version sets the flag to <code>false</code> on all other plugins with the same key. The promoted version is the one resolved by an indirect reference of the form <code>\$/plugins/<key></code> or a plugin name argument without a specified version.
<code>publishArtifactVersion</code>	Publishes an artifact version to an artifact repository.
<code>removeDependentsFromArtifactVersion</code>	Removes a list of dependent artifact versions from an existing artifact version.
<code>removeResourcesFromPool</code>	Removes resources from a specified resource pool.
<code>removeUsersFromGroup</code>	Removes one or more users from a particular group.
<code>restoreAclInheritance</code>	Restores ACL inheritance at the given object.
<code>retrieveArtifactVersions</code>	Retrieves the most recent artifact version, including its dependents, from an artifact repository.

Commands	Description
<code>runProcedure</code>	Creates and starts a new job using a procedure directly or specified indirectly through a schedule.
<code>runWorkflow</code>	Runs the specified workflow definition, returns the workflow name.
<code>sendEmail</code>	Facilitates sending an email from the command-line or a Command Step without setting up an Email Notifier. This API is more dynamic than an email notifier because you do not need to setup some kind of a template beforehand. This API also makes sending email attachments easier than using a notifier template.
<code>setDatabaseConfiguration</code>	Sets the database configuration on the server. If the server is in bootstrap mode, these changes take effect immediately and the server attempts to start. If the server is running, these changes have no effect until the server is restarted.
<code>setJobName</code>	Sets the name of a running job.
<code>setProperty</code>	Sets the value for the specified property.
<code>shutdownServer</code>	Shuts down the ElectricCommander server.
<code>testDirectoryProvider</code>	Tests an LDAP directory provider.
<code>transitionWorkflow</code>	Manually transition from the workflow active state.
<code>uninstallPlugin</code>	Uninstalls a plugin, deleting the associated project and any installed files.

API commands - ACL Management

```
breakAclInheritance
checkAccess
createAclEntry
deleteAclEntry
getAccess
getAclEntry
modifyAclEntry
restoreAclInheritance
```

breakAclInheritance

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

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

Arguments	Descriptions
artifactName	The name of the artifact.
artifactVersionName	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name—the Commander server interprets either name form correctly.
credentialName	<code>credentialName</code> can be one of two forms: relative (for example, <code>"cred1"</code>) - the credential is assumed to be in the project that contains the request target object. A qualifying project name is required . absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>) - the credential can be from any specified project, regardless of the target object's project.
configName	The name of the email configuration.

Arguments	Descriptions
gatewayName	The name of the gateway.
groupName	The full name of the group. For Active Directory and LDAP, the full name if the full DN.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.
pluginName	The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure or a path to a procedure, including the name. Also requires <code>projectName</code>
projectName	The name of the project - may be a path. The project name is ignored for credentials, procedures, steps, and schedules if they are specified as a path.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository used for artifact management.
resourceName	The name of a resource.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of a schedule - may be a path to a schedule. Also requires <code>projectName</code>
stateDefinitionName	The name of the state definition.

Arguments	Descriptions
stateName	The name of the state.
stepName	The name of the step - may be a path to the step. Also requires projectName and procedureName
systemObjectName	System objects names include: admin artifactVersions directory emailConfigs log plugins server session workspaces
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The full name of a user (for Active Directory or LDAP, this may be user@domain).
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of a workspace.
zoneName	The name of the zone.

Positional arguments

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

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool breakAclInheritance ...

Example

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

[Back to Top](#)

checkAccess

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

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

Arguments	Descriptions
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact container of the property sheet which owns the property.
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>
<code>componentName</code>	The name of the component container of the property sheet which owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	<p>The name of the credential container of the property sheet which owns the property.</p> <p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, <code>"cred1"</code>) - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.</p> <p>absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>) - the credential can be from any specified project, regardless of the target object's project.</p>

Arguments	Descriptions
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
gatewayName	The name of the gateway container of the property sheet.
groupName	The full name of the group container of the property sheet which owns the property. For Active Directory and LDAP, this is a full DN.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier (UUID) for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.
path	Property path string.
pluginName	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure - may be a path to the procedure. Also requires <code>projectName</code>
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.

Arguments	Descriptions
projectName	<p>The name of the project container of the property sheet which owns the property that may be a path; must be unique among all projects.</p> <p>The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.</p>
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	<p>The name of the schedule - may be a path to the schedule.</p> <p>Also requires projectName</p>
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	<p>The name of the step - may be a path to the step.</p> <p>Also requires projectName and procedureName</p>
systemObjectName	<p>System object names include:</p> <p>admin directory licensing log plugins priority projects </p>
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The full name of the user. For Active Directory and LDAP, the name may be user@domain.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace.
zoneName	The name of the zone.

Positional arguments

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

Response

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

ec-perl

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

Example

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

ectool

syntax: `ectool checkAccess ...`

Example

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

[Back to Top](#)

createAclEntry

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

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

Arguments	Descriptions
<code>artifactName</code>	The name of the artifact.
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>

Arguments	Descriptions
credentialName	<p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
configName	The name of the email configuration.
gatewayName	The name of the gateway.
groupName	The name of a group.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by findObjects and getObjects.
pluginName	The name of the plugin - the plugin key for a promoted plugin or plugin key and version for an unpromoted plugin.
principalName	This is either a user or a group name.
principalType	This is either user or group.
Privileges: readPrivilege modifyPrivilege executePrivilege changePermissionsPrivilege	<allow deny> If a privilege is not specified, permission is set to inherit from its parent object's ACL.
procedureName	The name of the procedure. Also requires projectName

Arguments	Descriptions
projectName	The name of the project.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step. Also requires projectName and procedureName
systemObjectName	System object names include: admin artifacts directory emailConfigs forceAbort licensing log plugins priority projects repositories resources server session test workspaces zonesAndGateways
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The full name of the user.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace.
zoneName	The name of the zone.

Positional arguments

principalType,principalName

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool createAclEntry <principalType> <principalName> ...

Example

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

[Back to Top](#)

deleteAclEntry

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

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

Arguments	Descriptions
artifactName	The name of the artifact.
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question.</p> <p>This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.</p>

Arguments	Descriptions
credentialName	<p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
configName	The name of the email configuration.
gatewayName	The name of the gateway.
groupName	The name of a group whose ACL entry you want to delete.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
objectId	An object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
notifierName	The name of the email notifier whose ACL entry you want to delete.
pluginName	The name of the plugin whose ACL entry you want to delete.
principalName	This is either the user or the group name.
principalType	This is either a user or a group <user group>. Defaults to "user".
procedureName	<p>The name of the procedure whose ACL entry you want to delete.</p> <p>Also requires <code>projectName</code>, where this procedure is a member.</p>
projectName	The name of the project where you are deleting an ACL entry.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.

Arguments	Descriptions
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource whose ACL entry you want to delete.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule whose ACL entry you want to delete. Also requires projectName from which this schedule runs procedures.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step whose ACL entry you want to delete. Also requires projectName and procedureName to indicate where this step resides.
systemObjectName	System object names include: admin directory licensing log plugins priority projects resources server session workspaces
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user who's ACL entry you want to delete.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace whose ACL entry you want to delete.
zoneName	The name of the zone.

Positional arguments

principalType, principalName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

getAccess

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

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

Arguments	Descriptions
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the credential container of the property sheet which owns the property. The name of the artifact.
artifactVersionName	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.

Arguments	Descriptions
credentialName	<p>The name of the credential container of the property sheet which owns the property. <code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p>
emulateRestoreInheritance	<p>Whether or not to include one level of broken inheritance if it exists. Used for seeing what access would look like if the lowest level of broken inheritance was restored.</p> <p><Boolean flag - 0 1 true false> If set to 1, this argument returns ACL information to what it would be if inheritance were restored on this object.</p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
gatewayName	The name of the gateway container of the property sheet.
groupName	The name of the group container of the property sheet that owns the property.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier that contains the ACL.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
path	Property path string.

Arguments	Descriptions
pluginName	The name of the plugin that contains the ACL.
procedureName	The name of the procedure containing the ACL. Also requires projectName
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.
projectName	The name of the project that contains the ACL; must be unique among all projects.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource that contains the ACL.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the ACL. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the ACL. Also requires projectName and procedureName
systemObjectName	System objects include: admin artifactVersions directory emailConfigs log plugins server session workspaces
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user that contains the ACL.

Arguments	Descriptions
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace that contains the ACL.
<code>zoneName</code>	The name of the zone.

Positional arguments

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

Response

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

ec-perl

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

Example

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

ectool

syntax: `ectool getAccess ...`

Example

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

[Back to Top](#)

getAclEntry

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

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

Arguments	Descriptions
<code>artifactName</code>	The name of the artifact.

Arguments	Descriptions
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "<code>groupId:artifactKey:version</code>" and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>
credentialName	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p>
configName	The name of the email configuration.
gatewayName	The name of the gateway.
groupName	The name of the group.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin - the plugin key for a promoted plugin or plugin key and version for an unpromoted plugin.
principalName	This is either the user or group name.
principalType	This is either <code>user</code> or <code>group</code> .

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure. Also requires <code>projectName</code>
<code>projectName</code>	The name of the project.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code>
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The full name of the user.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace.
<code>zoneName</code>	The name of the zone.

Positional arguments

`principalType`, `principalName`

Response

One `aclEntry` element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

modifyAclEntry

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

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

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

Arguments	Descriptions
<code>artifactName</code>	The name of the artifact.
<code>artifactVersionName</code>	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.

Arguments	Descriptions
credentialName	<p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
configName	The name of the email configuration.
gatewayName	The name of the gateway.
groupName	The name of the group containing the ACL entry.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier containing the ACL entry.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin containing the ACL entry.
Privileges: readPrivilege modifyPrivilege executePrivilege changePermissionsPrivilege	<allow deny>
principalName	This is either the user or group name.
principalType	This is either <code>user</code> or <code>group</code> .
procedureName	<p>The name of the procedure containing the ACL entry.</p> <p>Also requires <code>projectName</code></p>
projectName	The name of the project containing the ACL entry.

Arguments	Descriptions
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the ACL entry.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the ACL entry. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the ACL entry. Also requires projectName and procedureName
systemObjectName	System object names include: admin artifacts directory emailConfigs forceAbort licensing log plugins priority projects repositories resources server session test workspaces zonesAndGateways
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user containing the ACL entry.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the ACL entry.
zoneName	The name of the zone.

Positional arguments

principalType, principalName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

restoreAclInheritance

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

Note: You must use object locators to specify the object where you want to restore ACL inheritance.

Arguments	Descriptions
artifactName	The name of the artifact.
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question.</p> <p>This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>

Arguments	Descriptions
credentialName	<p>The name of the credential whose ACL inheritance you want to restore.</p> <p>credentialName can be one of two forms:</p> <p>relative for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Also requires projectName</p>
configName	The name of the email configuration.
gatewayName	The name of the gateway.
groupName	The name of the group whose ACL inheritance you want to restore.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	<p>The name of the email notifier whose ACL inheritance you want to restore.</p> <p>Also requires projectName and procedureName; projectName, procedureName, and stepName; jobId or jobStepId</p>
objectId	This is an object identifier returned by findObjects and getObjects.
pluginName	The name of the plugin whose ACL inheritance you want to restore.
procedureName	<p>The name of the procedure whose ACL inheritance you want to restore.</p> <p>Also requires projectName</p>

Arguments	Descriptions
<code>projectName</code>	The name of the project whose ACL inheritance you want to restore.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource whose ACL inheritance you want to restore.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule whose ACL inheritance you want to restore. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step whose ACL inheritance you want to restore. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	The name of the system object whose ACL inheritance you want to restore. System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code>
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The name of the user whose ACL inheritance you want to restore.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace whose ACL inheritance you want to restore.
<code>zoneName</code>	The name of the zone.

Positional arguments

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

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool restoreAclInheritance ...

Example

```
ectool restoreAclInheritance --projectName "Sample Project"
```

[Back to Top](#)

API commands - Applications

```
createApplication
deleteApplication
getApplication
getApplications
modifyApplication
```

createApplication

Creates a new application for a project.

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

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

Response

Returns an application element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

deleteApplication

Delete an application.

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

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

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool deleteApplication default appToDelete
```

[Back to Top](#)

getApplication

Finds an application by name.

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

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

Response

Retrieves the specified application element.

ec-perl

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

Example

```
$ec->getApplication("Default", "newApp");
```

ectool

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

Example

```
ectool getApplication default newApp
```

[Back to Top](#)

getApplications

Retrieves all applications in a project.

You must specify the `projectName` argument.

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

Response

Retrieves zero or more application elements.

ec-perl

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

Example

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

ectool

syntax:ectool getApplications <projectName>

Example

```
ectool getApplications default
```

[Back to Top](#)

modifyApplication

Modifies an existing application.

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

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

Response

Retrieves an updated application element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

API commands - Application Tier

```
createApplicationTier
deleteApplicationTier
getApplicationTier
getApplicationTiersinComponent
modifyApplicationTier
```

createApplicationTier

Creates a new application tier in the application.

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

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

Response

Returns an application tier element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

deleteApplicationTier

Deletes a tier from an application.

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

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

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool deleteApplicationTier default newApp appTierToDelete
```

[Back to Top](#)

getApplicationTier

Finds an application tier by name.

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

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

Response

Retrieves an application tier element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getApplicationTier default newApp appTier1
```

[Back to Top](#)

getApplicationTiers

Retrieves all application tiers in an application.

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

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

Response

Retrieves zero or more application tier elements.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getApplicationTiers default newApp
```

[Back to Top](#)

getApplicationTiersInComponent

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

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

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

Response

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

ec-perl

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

Example

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

ectool

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

Example

```
ectool getApplicationTiersInComponent default newComponent
```

[Back to Top](#)

modifyApplicationTier

Modifies an existing tier in the application.

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

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

Response

Retrieves an updated application tier element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

API commands - Artifact Management

<code>addDependentsToArtifactVersion</code>	<code>getArtifactVersions</code>
<code>cleanupArtifactCache</code>	<code>getManifest</code>
<code>cleanupRepository</code>	<code>getRepositories</code>
<code>createArtifact</code>	<code>getRepository</code>
<code>createRepository</code>	<code>modifyArtifact</code>
<code>deleteArtifact</code>	<code>modifyArtifactVersion</code>
<code>deleteArtifactVersion</code>	<code>modifyRepository</code>
<code>deleteRepository</code>	<code>moveRepository</code>
<code>findArtifactVersions</code>	<code>publishArtifactVersion</code>
<code>getArtifact</code>	<code>removeDependentsFromArtifactVersion</code>
<code>getArtifacts</code>	<code>retrieveArtifactVersions</code>
<code>getArtifactVersion</code>	

addDependentsToArtifactVersion

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

You must specify an `artifactVersionName`.

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

Positional arguments

`artifactVersionName`

Response

None or status OK message.

ec-perl

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

Example

```
# Add a dependency on cmdr:SDK:1.2.0 and the most current version of core:infra tha
t
# is greater than or equal to 2.1.0.
```

```
$cmdr->addDependentsToArtifactVersion ({artifactVersionName => "myGroup:myAKey:1.0.
0-55",
dependentArtifactVersions => ["cmdr:SDK:1.2.0", "core:infra:[2.1.0,]"]});
```

ectool

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

Example

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

[Back to Top](#)

cleanupArtifactCache

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

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

You must specify a `cacheDirectory`.

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

Positional arguments

`cacheDirectory`

Response

Returns a list of directories that were deleted.

ec-perl

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

Example

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

ectool

syntax: `ectool cleanupArtifactCache <cacheDirectory>`

Example

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

[Back to Top](#)

cleanupRepository

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

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

You must specify a `backingStoreDirectory`.

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

Positional arguments

`backingStoreDirectory`

Response

Returns a list of directories that were deleted.

ec-perl

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

Example

```

use strict;
use ElectricCommander;

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

```

ectool

syntax: ectool cleanupRepository <backingStoreDirectory>

Example

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

[Back to Top](#)

createArtifact

Creates a new artifact.

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

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

Positional arguments

groupId, artifactKey

Response

Returns an [artifact](#) element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

createRepository

Creates a repository for one or more artifacts.

You must specify a repositoryName.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
repositoryDisabled	<Boolean flag -0 1 true false> Determines whether the repository is disabled. Default is "false".
repositoryName	The name of the artifact repository.
url	The URL to use to communicate with the repository server.
zoneName	The name of the zone where this repository resides.

Positional arguments

repositoryName

Response

Returns a [repository](#) element.

ec-perl

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

Example

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

ectool

syntax: ectool createRepository <repositoryName> ...

Example

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

[Back to Top](#)

deleteArtifact

Deletes an existing artifact element and all artifact versions.

You must specify an artifactName.

Arguments	Descriptions
artifactName	The name of the artifact to delete.

Positional arguments

artifactName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool deleteArtifact <artifactName>

Example

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

[Back to Top](#)

deleteArtifactVersion

Deletes artifact version metadata from the Commander database.

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

You must specify an `artifactVersionName`.

Arguments	Descriptions
<code>artifactVersionName</code>	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.

Positional arguments

`artifactVersionName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteArtifactVersion(<artifactVersionName>);`

Example

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

ectool

syntax: `ectool deleteArtifactVersion <artifactVersionName>`

Example

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

[Back to Top](#)

deleteRepository

Deletes artifact repository metadata from the Commander database.

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

You must supply a `repositoryName`.

Arguments	Descriptions
<code>repositoryName</code>	The name of the artifact repository to delete.

Positional arguments

`repositoryName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteRepository <repositoryName>`

Example

```
ectool deleteRepository cmdrReposOne
```

[Back to Top](#)

findArtifactVersions

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

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

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

Note: The `retrieveArtifactVersions` API uses this API to find the appropriate artifact version in the Commander server

and then retrieves the artifact version from a repository. You may prefer to use the

`retrieveArtifactVersions` API

instead of this API because while this API returns slightly different information, it also has the side-effect of

"retriever
step registration" mentioned above.

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

Arguments	Descriptions
filter	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p> <p>Two types of filters:</p> <p style="padding-left: 40px;">"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p style="padding-left: 40px;">"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.</p> <p>Property filter operators are:</p> <pre style="padding-left: 40px;">between (2 operands) contains (1) equals (1) greaterOrEqual (1) greaterThan (1) in (1) lessOrEqual (1) lessThan (1) like (1) notEqual (1) notLike (1) isNotNull (0) isNull (0)</pre> <p>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.</p> <p>Boolean operators are:</p>

Arguments	Descriptions
	<code>not</code> (1 operand) <code>and</code> (2 or more operands) <code>or</code> (2 or more operands)
<code>artifactKey</code>	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>artifactName</code>	The name of an artifact.
<code>artifactVersionName</code>	The name of an artifact version.
<code>groupId</code>	A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>includeDependents</code>	Options are: <ul style="list-style-type: none">• <code>0/false</code> – dependent artifacts are not retrieved.• <code>1/true</code> – dependent artifacts are retrieved.
<code>jobStepId</code>	The unique identifier for the job step (if any), that is making the request. This job step will be marked as a retriever for the matching artifact versions.
<code>versionRange</code>	The range of versions to search. Version range syntax is standard number interval notation. <code>()</code> marks exclusive ranges and <code>[]</code> marks inclusive ranges.

Positional arguments

None

Response

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

ec-perl

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

Example 1

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

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

Example 2

```
# Find the most current core:infra artifact version with QA approval level 3 or above.
$cmdr->findArtifactVersions({groupId => "core",
                           artifactKey => "infra",
                           filter => {propertyName => "qaLevel",
                                       operator => "greaterOrEqual",
                                       operand1 => "3"}});
```

ectool

Not supported.

[Back to Top](#)

getArtifact

Retrieves an artifact by name.

You must specify an `artifactName`.

Arguments	Descriptions
<code>artifactName</code>	The name of the artifact.

Positional arguments

`artifactName`

Response

Retrieves an [artifact](#) element.

ec-perl

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

Example

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

ectool

syntax: `ectool getArtifact <artifactName>`

Example

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

[Back to Top](#)

getArtifacts

Retrieves all artifacts in the system.

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

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [artifact](#) elements.

ec-perl

syntax: \$cmdr->getArtifacts ();

Example

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

ectool

syntax: ectool getArtifacts

Example

```
ectool getArtifacts
```

[Back to Top](#)

getArtifactVersion

Retrieves an artifact version by its name.

You must specify an artifactVersionName.

Arguments	Descriptions
<code>artifactVersionName</code>	The name of the artifact version to retrieve. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.
<code>includeRetrieverJobs</code>	<Boolean flag - 0 1 true false> If set to 1, this argument includes <code>jobId</code> and <code>jobName</code> in returned information. A retriever job is any job that has retrieved the artifact version.
<code>includeRetrieverJobSteps</code>	<Boolean flag - 0 1 true false> If set to 1, this argument includes <code>jobId</code> , <code>jobName</code> , and <code>jobStepId</code> information. A retriever job is any job that has retrieved the artifact version. Because there is no bound to how many job steps may retrieve a given artifact version, the server limits the response to the most recent 200 job steps.
<code>maxRetrievers</code>	If one of the <code>includeRetriever*</code> options are specified, return at most "this many" of the most recent retrievers. Without this option, the Commander server will return all retrievers.

Positional arguments

`artifactVersionName`

Response

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

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

getArtifactVersions

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

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

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

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

Positional arguments

None

Response

Zero or more [artifactVersion](#) elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getArtifactVersions ...`

Example

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

[Back to Top](#)

getManifest

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

You must specify the `artifactVersionName`.

Arguments	Descriptions
<code>artifactVersionName</code>	The name of the artifact version whose manifest you want to retrieve.

Positional arguments

None

Response

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

ec-perl

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

Example

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

ectool

syntax: `ectool getManifest <artifactVersionName>`

Example

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

getRepositories

Retrieves all artifact repository objects known to the Commander server.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [repository](#) elements.

ec-perl

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

Example

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

ectool

syntax: ectool getRepositories

Example

```
ectool getRepositories
```

[Back to Top](#)

getRepository

Retrieves an artifact repository by its name.

You must specify a repositoryName.

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

Positional arguments

repositoryName

Response

One [repository](#) element.

ec-perl

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

Example

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

ectool

syntax: ectool getRepository <repositoryName>

Example

```
ectool getRepository myRepository
```

[Back to Top](#)

modifyArtifact

Modifies an existing artifact.

You must specify an artifactName.

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

Positional arguments

artifactName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

modifyArtifactVersion

Modifies an existing artifact version.

You must specify an artifactVersionName.

Arguments	Descriptions
artifactVersionName	The name of the artifact version to modify. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name--the Commander server interprets either name form correctly.
artifactVersionState	The state of the artifact version. <publishing available unavailable>.
dependentArtifactVersions	One or more artifact version queries. The most current match for each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: <groupId>:<artifactKey>:<versionRange> (version range is optional). Note: The absence of this argument does not clear or modify the dependent artifact version list for this artifact version.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are: <a> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
newName	Any name you choose to use as the new name for this artifact version.
removeAllDependentArtifactVersions	<Boolean flag - 0 1 true false> Defaults to "false." Removes all dependent artifacts from this artifact version. Subsequent "retrieves" will no longer retrieve dependent artifacts for this artifact version.
repositoryName	The name of the artifact repository.

Positional arguments

artifactVersionName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool modifyArtifactVersion <artifactVersionName> ...

Example

```
ectool modifyArtifactVersion "myGroup:myKey:1.0.1-57385" --artifactVersionState una
vailable
```

[Back to Top](#)

modifyRepository

Modifies an existing artifact repository.

You must specify a `repositoryName`.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
newName	Supply any name of your choice to rename the repository.
repositoryDisabled	<Boolean flag - 0 1 true false> Marks the repository as enabled or disabled. If you do not supply this option, the state of the repository is unchanged.
repositoryName	The name of the artifact repository.
url	The URL used to communicate with the artifact repository.
zoneName	The name of the zone where this repository resides.

Positional arguments

`repositoryName`

Response

Returns a modified `repository` element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

moveRepository

Moves an artifact repository in front of another, specified repository or to the end of the list.

This API does not move artifact version data to another repository server machine. Only the repository order in which Commander searches to retrieve an artifact version is changed.

You must specify a `repositoryName`.

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

Positional arguments

`repositoryName`

Response

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

ec-perl

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

Example

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

ectool

syntax: ectool moveRepository <repositoryName> ...

Example

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

[Back to Top](#)

publishArtifactVersion

Publishes an artifact version to an artifact repository.

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

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

Arguments	Descriptions
<code>artifactKey</code>	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>artifactName</code>	The name of an artifact.
<code>compress</code>	<Boolean flag - 0 1 true false> Default is "true". Controls whether or not the artifact version is compressed during transport, which improves performance for cases where artifact version files are compressible, saving network bandwidth. Where artifact version files are not compressible, performance is reduced. Another consideration is that the artifact version is stored compressed/uncompressed based on this setting in the repository backing-store.
<code>dependentArtifactVersions</code>	One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: <code><groupId>:<artifactKey>:<versionRange></code> (<code>versionRange</code> is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.

Arguments	Descriptions
description	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
excludePatterns	<p>Semi-colon delimited list of file-path patterns indicating which files/directories under "fromDirectory" to exclude when publishing an artifact version. Defaults to "empty," which means no files are excluded. See more information on "pattern syntax" below.</p>
followSymlinks	<p><i><Boolean flag - 0 1 true false></i> Default is "true".</p> <p>If true, follow symbolic links and record the target file contents with the symbolic link name in the artifact. If false, record the symbolic link as a symbolic link. Following symbolic links causes the publish API to remain compatible with previous releases.</p>
fromDirectory	<p>The directory containing files to publish as the artifact version. A subset of files can be published based on <code>includePatterns</code> and <code>excludePatterns</code>.</p>
groupId	<p>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</p>
includePatterns	<p>Semi-colon delimited list of file-path patterns indicating which files/directories under "fromDirectory" to publish in the artifact version. Defaults to "empty," which means all files will be included. Conversely, if only two files are "included," no other files except those two will be included. See more information on "pattern syntax" below.</p>
repositoryName	<p>The name of the artifact repository where you want to publish.</p>

Arguments	Descriptions
version	<p>Unique identifier for the artifact version in the form: <code>major.minor.patch-qualifier-buildNumber</code> <code>major</code>, <code>minor</code>, <code>patch</code>, and <code>buildNumber</code> are integers and <code>qualifier</code> can contain any character except the following: <code>\:<> ?*/</code></p> <p>If a version argument is provided, but does not follow the above format, the version will be considered <code>0.0.0-<user-specified-version-arg>-0</code> implicitly.</p> <p>See examples below.</p>

Version number examples

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

Pattern syntax

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

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

? - matches a single character

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

** - matches any number of directory levels

Examples:

Use `*.txt` to match any `.txt` file in the top-level directory.

Use `/**/*.txt` to match any `.txt` file in any child directory.

Use `**/*.txt` to match any `.txt` file at any level.

Positional arguments

None

Response

One [artifactVersion](#) element.

ec-perl

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

Example

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

```
$cmdr->publishArtifactVersion({artifactName => "myGroup:myKey",
                                version => "1.0.0-55",
                                dependentArtifactVersion => ["cmdr:SDK:1.2.0", "core:infra:{2.
1,}"]});
```

ectool

syntax: ectool publishArtifactVersion ...

Example

```
ectool publishArtifactVersion --artifactName "myGroup:myKey" --version "1.0.0-55"
--dependentArtifactVersion "cmdr:SDK:1.2.0":"core:infra"
```

[Back to Top](#)

removeDependentsFromArtifactVersion

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

You must specify the `artifactVersionName`.

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

Positional arguments

artifactVersionName

Response

None or status OK message.

ec-perl

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

Example

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

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

ectool

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

Example

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

[Back to Top](#)

retrieveArtifactVersions

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

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

You must specify search criteria options to locate the artifact versions you want to retrieve.

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

Arguments	Descriptions
filters	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p> <p>Two types of filters:</p> <p>"property filters" are used to select objects based on the value of the object's intrinsic or custom property.</p> <p>"boolean filters" ("and", "or", "not") are used to combine one or more filters using boolean logic.</p> <p>Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.</p> <p>Property filter operators are:</p> <ul style="list-style-type: none"> between (2 operands) contains (1) equals (1) greaterOrEqual (1) greaterThan (1) in (1) lessOrEqual (1) lessThan (1) like (1) notEqual (1) notLike (1) isNotNull (0) isNull (0) <p>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property</p>

Arguments	Descriptions
	<p>filter or a boolean filter.</p> <p>Boolean operators are:</p> <ul style="list-style-type: none"> not (1 operand) and (2 or more operands) or (2 or more operands)
<code>groupId</code>	A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>includeDependents</code>	<p>Options are:</p> <ul style="list-style-type: none"> • 0/false – dependent artifacts are not retrieved. • 1/true – dependent artifacts are retrieved.
<code>overwrite</code>	<p>Options are:</p> <ul style="list-style-type: none"> • true – deletes previous content in the directory and replaces the content with your new version. • false – (existing behavior) if the directory does not exist, one will be created and filled with the artifact's content. If the directory exists, a new directory is created with a unique name and the artifact contents is supplied there. • update – this is similar to a merge operation—two artifact versions can be moved into the same directory, but individual files with the same name will be overwritten.
<code>repositoryNames</code>	<p>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.</p>

Arguments	Descriptions
<code>toDirectory</code>	<p>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.</p> <ul style="list-style-type: none"> • If the artifact version is in a local cache directory, it will be copied out of the cache. • If the artifact version is not in a cache directory, it will be downloaded directly to the specified directory, without putting it into a cache. <code>toDirectory</code> overrides <code>cacheDirectory</code> for downloads.
<code>versionRange</code>	<p>The range of versions to search. Version range syntax is standard number interval notation. <code>()</code> marks exclusive ranges and <code>[]</code> marks inclusive ranges.</p>

Positional arguments

None

Response

Returns one or more `artifactVersion` elements.

ec-perl

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

Examples

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

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

ectool

syntax: `ectool retrieveArtifactVersions ...`

Example

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

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

[Back to Top](#)

API commands - Component

[addComponentToApplicationTier](#)
[createComponent](#)
[deleteComponent](#)
[getComponent](#)
[getComponents](#)
[getComponentsinApplicationTier](#)
[modifyComponent](#)
[removeComponentFromApplicationTier](#)

addComponentToApplicationTier

Adds the given component to the given application tier.

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

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

Response

Returns the component and specified application tier elements.

ec-perl

Syntax:


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

Example:

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

ectool

Syntax:

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

Example:

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

[Back to Top](#)

createComponent

Creates a new component for a project.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project; must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>pluginName</code>	Name of the plugin. Argument Type: String
<code>applicationName</code>	(Optional) Name of an application to scope this component to. Argument Type: String
<code>credentialName</code>	(Optional) Name of a credential to attach to this component. Argument Type: String

Arguments	Descriptions
<code>description</code>	(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String

Response

Returns a version control component element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

[Back to Top](#)

deleteComponent

Deletes a component.

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

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

Arguments	Descriptions
<code>applicationName</code>	(Optional) The name of an application to which this component is scoped. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool deleteComponent default VCSComponent
```

[Back to Top](#)

getComponent

Finds a component by name.

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

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

Arguments	Descriptions
<code>applicationName</code>	(Optional) Name of an application to which this component is scoped. Argument Type: String

Response

Retrieves the specified component element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getComponent default VCScomponent
```

[Back to Top](#)

getComponents

Retrieves all components in a project.

You must specify the `projectName` argument.

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

Response

Retrieves zero or more component elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getComponents default
```

[Back to Top](#)

getComponentsinApplicationTier

Returns the list of components in an application tier.

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

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

Response

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

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getComponentsInApplicationTier default newApp appTier1
```

[Back to Top](#)

modifyComponent

Modifies an existing component.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project; must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>credential Name</code>	(Optional) Name of the credential. Argument Type: String
<code>description</code>	(Optional) Comment text describing this component; not interpreted at all by ElectricFlow. Argument Type: String
<code>newName</code>	(Optional) New name of the component. Argument Type: String

Response

Retrieves an updated component element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

[Back to Top](#)

removeComponentFromApplicationTier

Removes the given component from the given application tier.

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

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

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

[Back to Top](#)

API Commands - Credential Management

```
attachCredential
createCredential
deleteCredential
detachCredential
getCredential
getCredentials
getFullCredential
modifyCredential
```

attachCredential

Attaches a credential to a step or a schedule.

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

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

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

Positional arguments

projectName, credentialName

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

createCredential

Creates a new credential for a project.

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

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

Positional arguments

projectName, credentialName

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

deleteCredential

Deletes a credential.

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

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

Positional arguments

`projectName`, `credentialName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

detachCredential

Detaches a credential from an object.

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

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

Positional arguments

`projectName`, `credentialName`

Response

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

`NoSuchCredential` if the specified credential does not exist.

`NoSuchSchedule` if the specified schedule does not exist.

ec-perl

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

Examples

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

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

ectool

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

Examples

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

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

[Back to Top](#)

getCredential

Finds a credential by name.

You must specify `projectName` and `credentialName`.

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

Positional arguments

`projectName`, `credentialName`

Response

One `credential` element.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

getCredentials

Retrieves all credentials in a project.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing these credentials.

Arguments	Descriptions
usableOnly	< <i>Boolean flag</i> - 0 1 true false> If set to 1, only those credentials that the currently logged-in user has execute privileges for will be returned.

Positional arguments

projectName

Response

Zero or more [credential](#) elements.

ec-perl

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

Example

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

ectool

syntax: ectool getCredentials <projectName> ...

Example

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

[Back to Top](#)

getFullCredential

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

You must specify the `credentialName`.

Arguments	Descriptions
credentialName	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<i>cred1</i>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>") - the credential can be from any specified project, regardless of the target object's project.</p>

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

Positional arguments

credentialName

Response

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

ec-perl

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

Example

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

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

ectool

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

Example

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

[Back to Top](#)

modifyCredential

Modifies an existing credential.

You must specify `projectName` and `credentialName`.

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

Positional arguments

projectName, credentialName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

API Commands - Database Configuration

[getDatabaseConfiguration](#)

[setDatabaseConfiguration](#)

getDatabaseConfiguration

Retrieves the current database configuration.

Arguments	Descriptions
None	

Positional arguments

None

Response

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

ec-perl

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

Example

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

ectool

syntax: `ectool getDatabaseConfiguration`

Example

```
ectool getDatabaseConfiguration
```

setDatabaseConfiguration

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

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

ElectricCommander assigns default values to the following three arguments--these values are derived from information you supply for the arguments below. The values for these arguments can be viewed in the

XML Response for `getDatabaseConfiguration`. You should not need to change these values, but "customDatabase" arguments may be used to over-ride Commander default values. Contact Electric Cloud Customer Support for assistance with using these arguments:

```
customDatabaseDialect
customDatabaseDriver
customDatabaseUrl
```

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

Arguments	Descriptions
<code>preserveSessions</code>	<i><Boolean flag - 0 1 true false></i> If ignoring a server mismatch, default behavior invalidates all sessions. Setting this flag to "true" preserves all sessions, allowing the server to reconnect to running jobs. This option is used in combination with <code>ignoreServerMismatch</code> .
<code>userName</code>	The name of the user required to access the database.

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

API Commands - Directory Provider Management

```
createDirectoryProvider
deleteDirectoryProvider
getDirectoryProvider
getDirectoryProviders
modifyDirectoryProvider
moveDirectoryProvider
testDirectoryProvider
```

createDirectoryProvider

Creates a new Active Directory or LDAP directory provider.

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

Arguments	Descriptions
<code>commonGroupNameAttribute</code>	The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code> , which is not searchable.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>domainName</code>	The domain name from which Active Directory server(s) are automatically discovered.
<code>emailAttribute</code>	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.
<code>enableGroups</code>	<i><Boolean flag -0 1 true false></i> Determines whether or not to enable external groups for the directory provider. Defaults to "true".

Arguments	Descriptions
<code>fullUserNameAttribute</code>	The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.
<code>groupBase</code>	This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
<code>groupMemberAttributes</code>	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.
<code>groupMemberFilter</code>	This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
<code>groupNameAttribute</code>	The group record attribute that contains the name of the group.
<code>groupSearchFilter</code>	This LDAP query is performed in the context of the groups directory to enumerate group records.
<code>managerDn</code>	The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access. Note: This user does not need to be an admin user with modify privileges.
<code>managerPassword</code>	If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.
<code>providerName</code>	This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.
<code>providerType</code>	<ldap activedirectory>

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

Positional arguments

providerName, providerType, url

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool createDirectoryProvider <providerName> ...

Example

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

[Back to Top](#)

deleteDirectoryProvider

Deletes an Active Directory or LDAP directory provider.

You must specify a `providerName`.

Arguments	Descriptions
<code>providerName</code>	The name of the directory provider you want to delete.

Positional arguments

`providerName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool deleteDirectoryProvider <providerName>

Example

```
ectool deleteDirectoryProvider AD3
```

[Back to Top](#)

getDirectoryProvider

Retrieves a directory provider by name.

You must specify a `providerName`.

Arguments	Descriptions
<code>providerName</code>	The name of the directory provider.

Positional arguments

`providerName`

Response

One `directoryProvider` element.

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

ec-perl

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

Example

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

ectool

syntax: `ectool getDirectoryProvider <providerName>`

Example

```
ectool getDirectoryProvider AD3
```

[Back to Top](#)

getDirectoryProviders

Retrieves all directory providers.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `directoryProvider` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getDirectoryProviders`

Example

```
ectool getDirectoryProviders
```

[Back to Top](#)

modifyDirectoryProvider

Modifies an existing LDAP directory provider.

You must specify the `providerName`.

Arguments	Descriptions
<code>commonGroupNameAttribute</code>	The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code> , which is not searchable.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>domainName</code>	The domain from which Active Directory servers are automatically discovered.
<code>emailAttribute</code>	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.

Arguments	Descriptions
<code>enableGroups</code>	<Boolean flag - 0 1 true false> Determines whether or not to enable external groups for the directory provider. Defaults to "true".
<code>fullUserNameAttribute</code>	The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.
<code>groupBase</code>	This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
<code>groupMemberAttributes</code>	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.
<code>groupMemberFilter</code>	This LDAP query is performed in the group directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed two parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
<code>groupNameAttribute</code>	The group record attribute that contains the name of the group.
<code>groupSearchFilter</code>	A filter name: this LDAP query is performed in the context of the groups directory to enumerate group records.
<code>managerDn</code>	The DN of a user who has read access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access. Note: This user does not need to be an admin user with modify privileges.
<code>managerPassword</code>	If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.

Arguments	Descriptions
<code>newName</code>	Supply any name of your choice to rename the directory provider.
<code>providerName</code>	This human readable name will be displayed in the user interface to identify users and groups that come from this provider.
<code>providerType</code>	<code><ldap activedirectory></code>
<code>realm</code>	This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The <code>realm</code> is appended to the user or group name when stored in the Commander server. For example, <code><user>@dir</code> (where the <code>realm</code> is set to "dir").
<code>url</code>	<p>The LDAP server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas.</p> <p>Note: Spaces in the <code>basedn</code> must be URL encoded (%20).</p>
<code>userBase</code>	This string is prepended to the <code>basedn</code> to construct the directory DN that contains user records.
<code>userNameAttribute</code>	The attribute in a user record that contains the user's account name.
<code>userSearchFilter</code>	This LDAP query is performed in the context of the user directory to search for a user by account name. The string " <code>{0}</code> " is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
<code>userSearchSubtree</code>	<Boolean flag - 0 1 true false> If "true", recursively search the subtree below the user base.
<code>useSSL</code>	<Boolean flag - 0 1 true false> Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true".

Positional arguments

`providerName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool modifyDirectoryProvider AD3 --emailAttribute email
```

[Back to Top](#)

moveDirectoryProvider

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

You must specify a `providerName`.

Arguments	Descriptions
<code>providerName</code>	The name of the directory provider to move.
<code>beforeProviderName</code>	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.

Positional arguments

`providerName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool moveDirectoryProvider <providerName> ...

Example

```
ectool moveDirectoryProvider AD3 --beforeProviderName AD2
```

[Back to Top](#)

testDirectoryProvider

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

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

Arguments	Descriptions
<code>commonGroupNameAttribute</code>	The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code> , which is not searchable.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>domainName</code>	The domain from which Active Directory servers are automatically discovered.
<code>emailAttribute</code>	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.
<code>enableGroups</code>	<Boolean flag - 0 1 true false> Determines whether or not to enable external groups for the directory provider. Defaults to "true".
<code>fullUserNameAttribute</code>	The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.

Arguments	Descriptions
<code>groupBase</code>	This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
<code>groupMemberAttributes</code>	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.
<code>groupMemberFilter</code>	<p>This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member.</p> <p>Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed two parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.</p>
<code>groupNameAttribute</code>	The group record attribute that contains the name of the group.
<code>groupSearchFilter</code>	This LDAP query is performed in the context of the groups directory to enumerate group records.
<code>managerDn</code>	<p>The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access.</p> <p>Note: This user does not need to be an admin user with modify privileges.</p>
<code>managerPassword</code>	If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.
<code>password</code>	The password for the user that you are testing for this provider. The command will prompt for the password if it is omitted.
<code>providerType</code>	<ldap activedirectory>

Arguments	Descriptions
realm	This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <code><user>@dir</code> (where the realm is set to "dir").
url	The LDAP server URL is in the form <code>protocol://host:port/basedn</code> . Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code> , 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas. Note: Spaces in the <code>basedn</code> must be URL encoded (<code>%20</code>).
useDefaults	<i><Boolean flag - 0 1 true false></i> If "true", defaults will be used for all fields not specified.
userBase	This string is prepended to the base DN to construct the directory DN that contains user records.
userName	The name of the user you are testing for this provider.
userNameAttribute	The attribute in a user record that contains the user's account name.
userSearchFilter	A filter name. This LDAP query is performed in the context of the user directory to search for a user by account name. The string " <code>{0}</code> " is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
userSearchSubtree	<i><Boolean flag - 0 1 true false></i> If "true", recursively search the subtree below the user base.
useSSL	<i><Boolean flag - 0 1 true false></i> Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true".

Positional arguments

`userName, password`

Response

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

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

API Commands - Email Configuration Management

```
createEmailConfig  
deleteEmailConfig  
getEmailConfig  
getEmailConfigs  
modifyEmailConfig
```

createEmailConfig

Creates a new email configuration.

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

Arguments	Descriptions
<code>configName</code>	The name of your email configuration.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>mailFrom</code>	The email address used as the email sender address for notifications.
<code>mailHost</code>	The name of the email server host.
<code>mailPort</code>	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.
<code>mailProtocol</code>	This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
<code>mailUser</code>	This can be an individual or a generic name like "Commander" - name of the email user on whose behalf Commander sends email notifications.
<code>mailUserPassword</code>	Password for the email user who is sending notifications.

Positional arguments

configName

Response

None or status OK message.

ec-perl

syntax: \$cmdr->createEmailConfig(<configName>, {<optionals>});

Example

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

ectool

syntax: ectool createEmailConfig <configName> ...

Example

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

[Back to Top](#)

deleteEmailConfig

Deletes an email configuration.

You must specify a configName.

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

Positional arguments

configName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteEmailConfig <configName>`

Example

```
ectool deleteEmailConfig emailA
```

[Back to Top](#)

getEmailConfig

Retrieves an email configuration by name.

You must specify a `configName`.

Arguments	Descriptions
<code>configName</code>	The name of the email configuration.

Positional arguments

`configName`

Response

Returns one `emailConfig` element.

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

ec-perl

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

Example

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

ectool

syntax: `ectool getEmailConfig <configName>`

Example

```
ectool getEmailConfig EmailConfig_test
```

[Back to Top](#)

getEmailConfigs

Retrieves all email configurations.

Arguments	Descriptions
None	

Positional arguments

None

Response

Returns one or more `emailConfig` elements.

Notes:

1. The `mailUserPassword` attribute value is not returned or displayed by the `getEmailConfigs` and `getEmailConfig` commands for security reasons.
2. The `configIndex` attribute is managed internally by ElectricCommander and cannot be used in any of the email configuration APIs. It is used internally to identify the order of `emailConfig` objects within the list.

ec-perl

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

Example

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

ectool

syntax: `ectool getEmailConfigs`

Example

```
ectool getEmailConfigs
```

[Back to Top](#)

modifyEmailConfig

Modifies an existing email configuration.

You must specify the `configName`.

Arguments	Descriptions
configName	The name of your email configuration.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
mailFrom	The email address used as the email "sender" address for notifications.
mailHost	The name of the email server host.
mailPort	The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTP). Specify a value for this argument when a non-default port is used.
mailProtocol	This is either SSMTP or SMTP (not case-sensitive). Default is SMTP.
mailUser	The name of the email user, which can be an individual or a generic name like "Commander".
mailUserPassword	The password for the email user.
newName	Supply any name of your choice to rename the email configuration.

Positional arguments

configName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool modifyEmailConfig <configName> ...

Example

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

[Back to Top](#)

API Commands - Email Notifier Management

```
createEmailNotifier
deleteEmailNotifier
getEmailNotifier
getEmailNotifiers
modifyEmailNotifier
sendEmail
```

createEmailNotifier

Creates an email notifier attached to the specified object.

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

Arguments	Descriptions
<code>condition</code>	Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
<code>configName</code>	If specified, this argument must specify the name of an <code>emailConfig</code> object. If not specified, the default value is the name of the FIRST <code>emailConfig</code> object defined for the Commander server (<code>emailConfig</code> objects are "ordered" Commander entities). Note: If using this argument, you must include either the <code>formattingTemplate</code> or the <code>formattingTemplateFile</code> argument also.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>destinations</code>	A mandatory argument for a <code>create</code> operation. A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.

Arguments	Descriptions
eventType	<p><onStart onCompletion></p> <p>"onStart" triggers an event when the job or job step begins.</p> <p>"onCompletion" triggers an event when the job finishes, no matter how it finishes. Default is "onCompletion."</p>
formattingTemplate	<p>This argument specifies a template for formatting email messages when an event [notification] is triggered by the emailNotifier. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.</p>
formattingTemplateFile	<p>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol.</p> <p>Contents of the <i>formatting template file</i> is read and stored in the "formatting template" field. This is an alternative argument for --formattingTemplate and is useful if the "formatting template" field spans multiple lines.</p>
jobId	<p>The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</p>
jobStepId	<p>The unique identifier for a job step, assigned automatically when the job step is created.</p>
notifierName	<p>The name of the email notifier.</p>
projectName	<p>The name of the project. Also requires procedureName</p>
procedureName	<p>The name of the procedure. Also requires projectName</p>
stateDefinitionName	<p>The name of the state definition.</p>
stateName	<p>The name of the state.</p>
stepName	<p>The name of the step. Also requires projectName and procedureName</p>
workflowDefinitionName	<p>The name of the workflow definition.</p>
workflowName	<p>The name of the workflow.</p>

Positional arguments

notifierName

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool createEmailNotifier <notifierName> ...

Example

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

[Back to Top](#)

deleteEmailNotifier

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

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

Arguments	Descriptions
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier you want to delete.
procedureName	The name of the procedure that contains this email notifier. Also requires projectName
projectName	The name of the project that contains this email notifier.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step that contains this email notifier. Also requires projectName and procedureName
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

notifierName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool deleteEmailNotifier <notifierName> ...

Example

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

[Back to Top](#)

getEmailNotifier

Retrieves an email notifier from a property sheet container.

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

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created.
<code>notifierName</code>	The name of your email notifier.
<code>procedureName</code>	The name of the procedure. Also requires the <code>projectName</code>
<code>projectName</code>	The name of the project that contains this email notifier. Also requires the <code>procedureName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step. Also requires the <code>procedureName</code> and the <code>projectName</code>
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.

Positional arguments

`notifierName`

Response

Returns one `emailNotifier` element.

ec-perl

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

Example

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

ectool

syntax: ectool getEmailNotifier <notifierName> ...

Example

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

[Back to Top](#)

getEmailNotifiers

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

You must specify one or more object locators.

Arguments	Descriptions
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
procedureName	The name of the procedure containing the email notifier. Also requires the projectName
projectName	The name of the project containing the email notifier. Also requires the procedureName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step. Also requires the procedureName and the projectName
workflowDefinitionName	The name of the workflow definition.

Arguments	Descriptions
<code>workflowName</code>	The name of the workflow.

Positional arguments

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

Response

Returns one or more `emailNotifier` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getEmailNotifiers ...`

Example

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

[Back to Top](#)

modifyEmailNotifier

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

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

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

You must specify a `notifierName`.

Arguments	Descriptions
<code>condition</code>	Only send mail if the condition evaluates to "true ". The condition is a string subject to property expansion. Notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is <i>a/ways</i> sent.

Arguments	Descriptions
configName	<p>If specified, this argument must specify the name of an <code>emailConfig</code> object. If not specified, the default value is the name of the FIRST <code>emailConfig</code> object defined for the Commander server (<code>emailConfig</code> objects are "ordered" ElectricCommander entities).</p> <p>Note: If using this argument, you must include either <code>formattingTemplate</code> or <code>formattingTemplateFile</code> also (not both arguments).</p>
description	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
destinations	<p>A space-separated list of valid email addresses, email aliases, or ElectricCommander user names, or a string subject to property expansion that expands into such a list.</p> <p>Note: This argument is mandatory for the "create" operation.</p>
eventType	<p><code><onStart onCompletion></code> "onStart" triggers an event when the job/jobstep begins. "onCompletion" triggers an event when the job finishes, no matter how it finishes.</p> <p>Default is "onCompletion."</p>
formattingTemplate	<p>This argument specifies a template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code>. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.</p>
formattingTemplateFile	<p>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol.</p> <p>Contents of the <i>formatting template file</i> is read and stored in the "formatting template" field. This is an alternative argument for <code>formattingTemplate</code> and is useful if the "formatting template" field spans multiple lines.</p>
jobId	<p>The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</p>

Arguments	Descriptions
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
newName	Supply any name of your choice to rename the email notifier.
notifierName	The name of your email notifier.
procedureName	The name of the procedure. Also requires projectName
projectName	The name of the project. Also requires the procedureName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step. Also requires the procedureName and the projectName
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

notifierName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool modifyEmailNotifier <notifierName> ...

Example

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

[Back to Top](#)

sendEmail

Facilitates sending an email from the command-line or a Command Job Step without setting up an Email Notifier.

This API is more dynamic than an email notifier because you do not need to setup some kind of a template beforehand. This API also makes sending email attachments easier than using a notifier template.

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

These options are: `mailHost`, `mailPort`, `mailFrom`, `mailUser`, and `mailUserPassword`.

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

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

Arguments	Descriptions
<code>configName</code>	The name of the email configuration to use. If no configuration is specified, the configuration named "default" will be used. Note: The user must have "execute" permission on the configuration.
<code>subject</code>	The subject of the email message.
<code>to</code>	A "To" recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.
<code>cc</code>	A "Cc" recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.
<code>bcc</code>	A "Bcc" recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.

Arguments	Descriptions
header	An RFC822 email header line (for example: "reply-to: user@host.com"). This option can be specified multiple times.
html	The body of a simple HTML message.
htmlFile	Reads the specified client-side file and uses it as the body of a simple HTML message.
text	The body of a simple text message.
textFile	Reads the specified client-side file and uses it as the body of a simple text message.
raw	A raw email message including headers to use as the basis for the email message. Additional options can be applied to this message. The value should be a properly formatted RFC822 message.
rawFile	Reads the specified client-side file and uses it as the entire mail message, including headers.
attachment	One or more client-side files to send as attachments. The filename extension is examined to determine the content-type. This option can be specified multiple times.
inline	<p><code><contentId>=<fileName> [<contentId>=<fileName> ...]</code></p> <p>One or more inline attachments specified as a <code>contentId</code> and a client-side filename. The filename extension is examined to determine the content-type. The <code>contentId</code> can be referenced in an HTML body using the <code>cid:protocol</code>.</p> <p>For example:</p> <p><code></code> could reference <code>--inlinemyImage=image.jpg</code></p> <p>This option can be specified multiple times.</p>
mailFrom	The "From" header to use when sending mail. Overrides the value from the email configuration if specified.
mailHost	The name of the mail server to use if no <code>configName</code> is specified. Overrides the value from the email configuration if specified.

Arguments	Descriptions
<code>mailPort</code>	The mail server port to use if no <code>configName</code> is specified. Overrides the value from the email configuration if specified.
<code>mailProtocol</code>	The mail protocol. Must be either SMTP or SMTPS. Overrides the value from the email configuration if specified.
<code>mailUser</code>	The user account to use when authenticating to the mail server. Overrides the value from the email configuration if specified.
<code>mailUserPassword</code>	The password to use when authenticating to the mail server. Overrides the value from the email configuration if specified.
<code>multipartMode</code>	<p><none mixed related mixedRelated></p> <p>Sets the multipart mode. Must be one of the following allowed multipart modes:</p> <p><code>none</code> - non-multipart message</p> <p><code>mixed</code> - single-root multipart element of type "mixed". Texts, inline elements, and attachments will be all be added to this root element.</p> <p><code>related</code> - multipart message with a single root multipart element of type "related". Texts, inline elements, and attachments will be added to this root element. Works on most mail clients, except Lotus Notes.</p> <p><code>mixedRelated</code> - multipart element "mixed" plus a nested multipart element of type "related". Texts and inline elements will be added to the nested "related" element, while attachments will be added to the "mixed" root element. Works on most mail clients other than Mac Mail and some situations on Outlook. If you experience problems, try "related".</p> <p>Note: <code>multipartMode</code> defaults to <code>none</code> unless there are multiple parts, in which case it defaults to <code>mixedRelated</code>. If both <code>text</code> and <code>html</code> arguments are specified, both values are sent as alternates in a multipart message.</p>

Positional arguments

None

Response

None or status OK message.

ec-perl

syntax: \$cmdr->sendEmail

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

Example

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

ectool

syntax: ectool sendEmail

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

Examples

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

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

[Back to Top](#)

API Commands - Environment Requests

[createEnvironment](#)
[createEnvironmentInventoryItem](#)
[deleteEnvironment](#)
[deleteEnvironmentInventoryItem](#)
[getEnvironment](#)
[getEnvironments](#)
[getEnvironmentApplications](#)
[getEnvironmentInventory](#)
[getEnvironmentInventoryItem](#)
[getEnvironmentInventoryItems](#)
[modifyEnvironment](#)
[modifyEnvironmentInventoryItem](#)

createEnvironment

Creates a new environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

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

Argument Type: String

Optional Arguments

`applicationName`

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

Argument Type: String

`applicationProjectName`

Description: Name of the application project.

Argument Type: String

description

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

Argument Type: String

environmentEnabled

Description: True to enable the environment.

Argument Type: Boolean

Response

Returns an environment element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

createEnvironmentInventoryItem

Creates a new environment inventory item.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

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

Argument Type: String

`componentName`

Description: Component that owns the inventory item.

Argument Type: String

`resourceName`

Description: Resource where the item is installed.

Argument Type: String

`artifactName`

Description: Artifact name for the inventory item.

Argument Type: String

`artifactVersion`

Description: Artifact version for the inventory item.

Argument Type: String

Optional Arguments

`artifactSource`

Description: Source of the artifact.

Argument Type: String

`artifactUrl`

Description: URL of the artifact.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted by ElectricCommander.

Argument Type: String

Response

Returns an environment inventory item.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

deleteEnvironment

Deletes an environment.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

```
ectool deleteEnvironment <projectName>  
  <environmentName>
```

Example:


```
ectool deleteEnvironment default envToDelete
```

deleteEnvironmentInventoryItem

Delete an inventory item.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

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

Argument Type: String

componentName

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

Argument Type: String

resourceName

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

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

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

ectool**Syntax:**

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

Example:

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

getEnvironment

Retrieves an environment by name.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

Optional Arguments

None

Response

Retrieves an environment element.

ec-perl**Syntax:**

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

Example:

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

ectool**Syntax:**

```
ectool getEnvironment <projectName> <environmentName>
```

Example:

```
ectool getEnvironment default newEnv
```

getEnvironments

Retrieves all environments in a project.

Required Arguments

`projectName`

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

Argument Type: String

Optional Arguments

None

Response

Retrieves zero or more environment elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

```
ectool getEnvironments <projectName>
```

Example:

```
ectool getEnvironments default
```

getEnvironmentApplications

Retrieves a list of applications installed on the given environment.

`projectName`

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

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

Optional Arguments

None

Response

Retrieves a list of applications for the specified environment.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getEnvironmentApplications default newEnv
```

getEnvironmentInventory

Retrieves a per-component grouped list of inventory items.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

Description: Name of the application.

Argument Type: String

Optional Arguments

None

Response

Retrieves a per-component grouped list of inventory items.

ec-perl

Syntax:

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

Example:

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

ectool**Syntax:**

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

Example:

```
ectool getEnvironmentInventory default newEnv App1
```

getEnvironmentInventoryItem

Retrieves an inventory item.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

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

Argument Type: String

componentName

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

Argument Type: String

resourceName

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

Argument Type: String

Optional Arguments

None

Response

Retrieves an inventory item.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

getEnvironmentInventoryItems

Retrieves all inventory items for a given environment.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

Optional Arguments

None

Response

Retrieves all inventory items for the specified environment.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

```
ectool getEnvironmentInventoryItems <projectName> <environmentName>
```

Example:

```
ectool getEnvironmentInventoryItems default newEnv
```

modifyEnvironment

Modifies an environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

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

Argument Type: String

Optional Arguments

`description`

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

Argument Type: String

`environmentEnabled`

Description: True to enable the environment.

Argument Type: Boolean

`newName`

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

Argument Type: String

Response

Retrieves an updated environment element.

ec-perl

Syntax:

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

Example:

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

ectool**Syntax:**

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

Example:

```
ectool modifyEnvironment default testEnv --newName modEnv  
--description exampleText
```

modifyEnvironmentInventoryItem

Modifies an existing environment inventory item.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

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

Argument Type: String

componentName

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

Argument Type: String

resourceName

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

Argument Type: String

artifactName

Description: Name of the artifact for the inventory item.

Argument Type: String

artifactVersion

Description: Version of the artifact for the inventory item.

Argument Type: String

Optional Arguments

artifactSource

Description: Source of the artifact.

Argument Type: String

artifactUrl

Description: URL of the artifact.

Argument Type: String

description

Description: Comment text describing this object; not interpreted by ElectricCommander.

Argument Type: String

Response

Retrieves an updated environment inventory item.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool modifyEnvironmentInventoryItem default testEnv App1 Component1 Server1  
Artifact1 V3
```

API Commands - Environment Tier

[createEnvironmentTier](#)

[deleteEnvironmentTier](#)

[getEnvironmentTier](#)

[getEnvironmentTiers](#)

[modifyEnvironmentTier](#)

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 ElectricCommander.

Argument Type: String

Response

Returns an environment tier element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool createEnvironmentTier default newEnv envTier1  
    --description exampleText
```

deleteEnvironmentTier

Deletes an environment tier.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

environmentTierName

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

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool deleteEnvironmentTier default newEnv tierToDelete
```

getEnvironmentTier

Retrieves an environment tier by name.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

environmentTierName

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

Argument Type: String

Optional Arguments

None

Response

Retrieves an environment tier element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getEnvironmentTier default newEnv envTier1
```

getEnvironmentTiers

Retrieves all environment tiers in an environment.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

Optional Arguments

None

Response

Retrieves zero or more environment tier elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getEnvironmentTiers default newEnv
```

modifyEnvironmentTier

Modifies an environment tier.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

environmentTierName

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

Argument Type: String

Optional Arguments

description

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

Argument Type: String

newName

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

Argument Type: String

Response

Retrieves an updated environment tier element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool modifyEnvironmentTier default newEnv envTier1  
    --description new_exampleText --newName envTierA
```

API Commands - Gateways/Zones Management

<code>createGateway</code>	<code>createZone</code>
<code>deleteGateway</code>	<code>deleteZone</code>
<code>getGateway</code>	<code>getZone</code>
<code>getGateways</code>	<code>getZones</code>
<code>modifyGateway</code>	<code>modifyZone</code>

createGateway

Creates a new gateway.

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

You must specify `gatewayName`.

Arguments	Descriptions
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>gatewayDisabled</code>	<Boolean flag - 0 1 true false > If set to 1, the gateway is disabled.
<code>gatewayName</code>	The name of the gateway. Supply any of your choice, but the name must be unique among other gateway names.
<code>hostName1</code>	The agent host name where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option is you want to use the host name from <i>Resource1</i> 's definition.
<code>hostName2</code>	The agent host name where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option is you want to use the host name from <i>Resource2</i> 's definition.

Arguments	Descriptions
port1	The port number used by <i>Resource1</i> - defaults to the port number used by the resource.
port2	The port number used by <i>Resource2</i> - defaults to the port number used by the resource.
resourceName1	The name of your choice for the first of two required gateway resources. Do not include "spaces" in a resource name.
resourceName2	The name of your choice for the second of two required gateway resources. Do not include "spaces" in a resource name.

Positional arguments

gatewayName

Response

Returns a gateway object.

ec-perl

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

Example

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

ectool

syntax: ectool createGateway <gatewayName> ...

Example

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

[Back to Top](#)

deleteGateway

Deletes a gateway.

You must supply a gatewayName.

Arguments	Descriptions
gatewayName	The name of the gateway to delete.

Positional arguments

gatewayName

Response

None

ec-perl

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

Example

```
$cmdr->deleteGateway ("AB_Gateway");
```

ectool

syntax: ectool deleteGateway <gatewayName>

Example

```
ectool deleteGateway "AB_Gateway"
```

[Back to Top](#)

getGateway

Finds a gateway by name.

You must specify a gatewayName.

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

Positional arguments

gatewayName

Response

Returns one [gateway](#) element.

ec-perl

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

Example

```
$cmdr->getGateway ("AB_Gateway");
```

ectool

syntax: ectool getGateway <gatewayName>

Example

```
ectool getGateway AB_Gateway
```

[Back to Top](#)

getGateways

Retrieves all gateways.

Arguments	Descriptions
None	

Positional arguments

None.

Response

Returns one or more [gateway](#) elements.

ec-perl

syntax: \$cmdr->getGateways();

Example

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

ectool

syntax: ectool getGateways

Example

```
ectool getGateways
```

[Back to Top](#)

modifyGateway

Modifies an existing gateway.

You must specify a `gatewayName`.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
gatewayDisabled	<Boolean flag - 0 1 true false > If set to 1, the gateway is disabled.
gatewayName	The name of the gateway you want to modify.
hostName1	The agent host name where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option is you want to use the host name from <i>Resource1</i> 's definition.
hostName2	The agent host name where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option is you want to use the host name from <i>Resource2</i> 's definition.
newName	Supply any name of your choice to rename the gateway.
port1	The port number used by <i>Resource1</i> - defaults to the port number used by the resource.
port2	The port number used by <i>Resource2</i> - defaults to the port number used by the resource.
resourceName1	The name of your choice for the first of two required gateway resources. Do not include "spaces" in a resource name.
resourceName2	The name of your choice for the second of two required gateway resources. Do not include "spaces" in a resource name.

Positional arguments

gatewayName

Response

An updated gateway object.

ec-perl

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

Example

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

ectool

syntax: ectool modifyGateway <gatewayName> ...

Example

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

[Back to Top](#)

createZone

Creates a new zone.

You must specify a `zoneName`.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are: <a> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
zoneName	The name of the zone. Supply any unique name of your choice.

Positional arguments

zoneName

Response

Returns a zone object.

ec-perl

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

Example

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

ectool

syntax: ectool createZone <zoneName> ...

Example

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

[Back to Top](#)

deleteZone

Deletes an existing zone.

You must specify a `zoneName`.

Arguments	Descriptions
zoneName	The name of the zone to delete.

Positional arguments

zoneName

Response

None

ec-perl

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

Example

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

ectool

syntax: ectool deleteZone <zoneName>

Example

```
ectool deleteZone DevZone
```

[Back to Top](#)

getZone

Finds a zone by name.

You must specify a `zoneName`.

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

Positional arguments

zoneName

Response

Returns a [zone](#) element, including a list of resources belonging to the zone.

ec-perl

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

Example

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

ectool

syntax: `ectool getZone <zoneName>`

Example

```
ectool getZone DevZone
```

[Back to Top](#)

getZones

Retrieves all zones.

Arguments	Descriptions
None	

Positional arguments

None

Response

Returns a zone object.

ec-perl

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

Example

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

ectool

syntax: ectool getZones

Example

```
ectool getZones
```

[Back to Top](#)

modifyZone

Modifies an existing zone.

You must specify a `zoneName`.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
newName	Supply any unique name of your choice to rename the zone.
zoneName	The name of this zone.

Positional arguments

zoneName

Response

Returns an updated zone element.

ec-perl

syntax: \$cmdr->modifyZone (<zoneName>, {...});

Example

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

ectool

syntax: ectool modifyZone <zoneName> ...

Example

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

[Back to Top](#)

API Commands - Job Management

<code>abortAllJobs</code>	<code>getJobsForSchedule</code>
<code>abortJob</code>	<code>getJobStatus</code>
<code>abortJobStep</code>	<code>getJobStepDetails</code>
<code>deleteJob</code>	<code>getJobStepStatus</code>
<code>findJobSteps</code>	<code>moveJobs</code>
<code>getJobDetails</code>	<code>runProcedure</code>
<code>getJobInfo</code>	<code>setJobName</code>
<code>getJobNotes</code>	
<code>getJobs</code>	

External Job APIs

<code>completeJob</code>	<code>modifyJob</code>
<code>completeJobStep</code>	<code>modifyJobStep</code>
<code>createJob</code>	
<code>createJobStep</code>	

abortAllJobs

Aborts all running jobs.

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

Positional arguments

None

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool abortAllJobs ...

Example

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

[Back to Top](#)

abortJob

Aborts a running job.

You must supply a `jobId`.

Arguments	Descriptions
<code>force</code>	<i><Boolean flag - 0 1 true false></i> If set to 1, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked "always run".
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>reason</code>	A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string - similar to a text Description "for your reference only."

Positional arguments

`jobId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool abortJob <jobId> ...

Example

```
ectool abortJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --force 1
```

[Back to Top](#)

abortJobStep

Aborts any type of step—command step or subprocedure step.

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

You must specify a `jobStepId`.

Arguments	Descriptions
<code>force</code>	<i><Boolean flag - 0 1 true false></i> If set to 1, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, for example, executing steps marked "always run".
<code>jobStepId</code>	The <code>jobStep</code> to abort - the unique identifier for a job step, assigned automatically when the job step is created.
<code>reason</code>	A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string - similar to a text Description "for your reference only."

Positional arguments

`jobStepId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

deleteJob

Deletes a job from the ElectricCommander database.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Positional arguments

`jobId`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteJob <jobId>`

Example

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

[Back to Top](#)

findJobSteps

Returns a list of job steps from a single job or from a single subprocedure job step. This API is used by the Job Details web page in the Commander UI. The elements in the list are returned in their natural "job order".

You must specify either a `jobId` or a `jobStepId`, but not both.

Arguments	Descriptions
<code>jobId</code>	The unique identifier for the job whose steps you want to retrieve - assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step whose job steps you want to retrieve - assigned automatically when the job step is created.
<code>filter</code>	A list of zero or more filter criteria definitions used to define objects to find. See the findObjects API for complete description for using filters.
<code>numObjects</code>	<full object count> This specifies the number of full job steps (not just the IDs) returned in the response. Returned job steps will be from the beginning of the list. If <code>numObjects</code> is not specified, all job steps in the list of object IDs are returned. Any and all job steps can be retrieved using the <code>getObjects</code> command.
<code>select</code>	This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example for <code>findObjects</code> for instructions on forming the list and passing it to the ElectricCommander Perl API.

Positional arguments

`jobId`, or `jobStepId`

Response

One or more [jobStep](#) elements.

ec-perl

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

Example 1

```
my $XPath = $cmdr->findJobSteps(
    {jobId    => "4fa765dd-73f1-11e3-b67e-b0a420524153",
     select  => [{propertyName => 'charEncoding'},
                 {propertyName => 'abc'}]});
print "Return data from Commander:\n" .
    $XPath-> findnodes_as_string("/"). "\n";
```

Example 2

```
my $XPath = $cmdr->findJobSteps({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153"});
```

```
print "Return data from Commander:\n" .
  $xPath-> findnodes_as_string("/"). "\n";
```

ectool

Not supported.

[Back to Top](#)

getJobDetails

Retrieves complete information about a job, including details from each job step.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>structureOnly</code>	<i><Boolean flag - 0 1 true false></i> Reduces the amount of information returned to minimal structural information.

Positional arguments

`jobId`

Response

One `job` element, including one or more `jobStep` elements.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getJobDetails 4fa765dd-73f1-11e3-b67e-b0a420524153 --structureOnly 1
```

[Back to Top](#)

getJobInfo

Retrieves all information about a job, **except** job step information.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Positional arguments

`jobId`

Response

One `job` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getJobInfo <jobId>`

Example

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

[Back to Top](#)

getJobNotes

Retrieves the notes property sheet from a job.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Positional arguments

`jobId`

Response

A [propertySheet](#) element that contains the job.

ec-perl

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

Example

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

ectool

syntax: `ectool getJobNotes <jobId>`

Example

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

[Back to Top](#)

getJobs

Retrieves summary information for a list of jobs. By default, all jobs are returned.

Notes:

1. If using `sortKey` or `sortOrder`, you must use both arguments together.
2. You can use `firstResult` and `maxResults` together or separately to select a limited sub-list of jobs for the result set.

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

Positional arguments

None

Response

One or more `job` elements. A `job` element contains summary information only.

ec-perl

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

Examples

How do I get the first 10 jobs (index 0-9)?

```
$cmdr-> getJobs ({maxResults=>10});
```

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

```
$cmdr-> getJobs ({firstResult=>10, maxResults=>10});
```

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

```
$cmdr-> getJobs ({sortBy=>'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 --sortBy start --sortOrder descending --maxResults 1
```

[Back to Top](#)

getJobsForSchedule

Retrieves jobs started by a specific schedule.

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

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

Positional arguments

projectName, scheduleName

Response

Returns an XML stream containing any number of `job` elements. The `job` elements contain summary information only.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getJobsForSchedule Test ea1
```

[Back to Top](#)

getJobStatus

Retrieves the status of a job.

You must specify the `jobId`.

Arguments	Descriptions
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Positional arguments

jobId

Response

Values for `status` and `outcome` as follows:

Possible values for `status`:

`pending` - The job is not yet runnable--it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.

`runnable` - The job is ready to run, but it is waiting for a resource to become available.

`running` - The job is assigned to a resource and is executing the step command.

`completed` - The job finished executing.

Possible values for `outcome`: The `outcome` is accurate only if the job status is "completed."

`success` - The job finished successfully.

`warning` - The job completed with no errors, but encountered some suspicious conditions.

`error` - The job has finished execution with errors.

ec-perl

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

Example

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

ectool

syntax: `ectool getJobStatus <jobId>`

Example

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

[Back to Top](#)

getJobStepDetails

Retrieves details for a job step.

You may never need to use this command. This information is available for all job steps in a job by using the `getJobDetails` command. The `getJobStepDetails` command can be used to refresh data for a single step if you need an update in real time.

You must specify `jobStepId`.

Arguments	Descriptions
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created.

Arguments	Descriptions
structureOnly	<Boolean flag - <0 1 true false> Reduces the amount of information returned to minimal structural information.

Positional arguments

jobStepId

Response

A [jobStep](#) element.

ec-perl

syntax: \$cmdr->getJobStepDetails(<jobStepId>, {...});

Example

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

ectool

syntax: ectool getJobStepDetails <jobStepId> ...

Example

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

[Back to Top](#)

getJobStepStatus

Retrieves the status of a job step.

You must specify the `jobStepId`.

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

Positional arguments

jobStepId

Response

A `status` tag - for example: <status>completed</status>

Possible values for `status`:

pending - The job step is not yet runnable--it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.

runnable - The job step is ready to run, but it is waiting for a resource to become available.

running - The job step is assigned to a resource and is executing the step command.

completed - The job step finished executing.

ec-perl

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

Example

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

ectool

syntax: `ectool getJobStepStatus <jobStepId>`

Example

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

[Back to Top](#)

moveJobs

Moves jobs from one project to another project.

You must specify `sourceProject` and `destinationProject`.

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

Positional arguments

`sourceProject, destinationProject`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->moveJobs (<sourceProject>, <destinationProject>);`

Example

```
$cmdr->moveJobs ("ProjectA", "ProjectB");
```

ectool

syntax: ectool moveJobs <sourceProject> <destinationProject> ...

Example

```
ectool moveJobs "ProjectA" "ProjectB"
```

[Back to Top](#)

runProcedure

Creates and starts a new job using a procedure directly or a procedure specified indirectly through a schedule. Returns a new job ID. If the `pollInterval` option is provided, wait until the job completes up to a maximum of `timeout` seconds (if also provided). If the `scheduleName` option is provided, the parameters provided by that schedule will be used.

runProcedure credentials - two types of credentials can be passed to `runProcedure`:

- Impersonation credentials
- Credential parameters

Impersonation credentials

Impersonation credentials are used to set the top level impersonation credential for a job. If specified, the impersonation credential [on the job] is used as the default impersonation credential for all steps in the job.

The impersonation credential can be specified in two ways. If the `credentialName` argument is supplied, the job looks for the named credential specified. If the user has execute permission on the specified credential,

`runProcedure` is allowed to start the job.

If the `userName` and `password` arguments are supplied, the job creates a transient credential to contain the pair. The transient credential is used by the job and then discarded when the job completes.

Only one of `credentialName` or `userName` should be specified. If both are specified, only `userName` is used.

Neither can be specified if the procedure being run already has a credential defined on the procedure or the project.

Credential parameters

If the procedure defines one or more credential parameters, `runProcedure` must specify a credential to use for each parameter. The `actualParameter` argument identifies the credential name to use for the parameter, and the `credential` argument specifies the user name for each defined credential. For each credential specified, ectool prompts for a password.

For example, for a procedure named 'proc1' with a single credential parameter named 'param1'. The following command could be used to pass a transient credential where the user name is 'joe' and the password

is 'plumber':

```
$ ectool runProcedure test --procedureName proc1 \
  --actualParameter param1=cred1 --credential cred1=joe
  cred1 password: plumber
```

Multiple parameters or credentials can be specified by having additional *name=value* pairs after the `actualParameter` or `credential` arguments. The same credential can be specified as the value for more than one actual parameter.

You must specify a `projectName` and either a `procedureName` or a `scheduleName`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Used in conjunction with <code>procedureName</code> to set the value of the actual parameters. Do not use this argument with <code>scheduleName</code> .
<code>credential</code>	Use the following syntax to specify a credential: <credName>=<userName> [<credName>=<userName> ...]
<code>credentialName</code>	<code>credentialName</code> can be one of two forms: relative (for example, " <i>cred1</i> ") - the credential is assumed to be in the project that contains the request target object. absolute (for example, " <i>/projects/BuildProject/credentials/cred1</i> ") - the credential can be from any specified project, regardless of the target object's project.
<code>destinationProject</code>	This argument is used to specify the name of the destination project.
<code>password</code>	The password for the user running the procedure.
<code>priority</code>	Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. Priority values are: <code>low normal high highest</code>

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure you want to run.
<code>projectName</code>	The name of the project that contains the procedure you want to run.
<code>scheduleName</code>	The name of the schedule. Use this option if you want to use the parameters from an existing specific schedule.
<code>userName</code>	The name of the user who is running the procedure.
Note: The following two arguments are used to control whether <code>runProcedure</code> returns immediately or waits until the job completes. *** If <code>pollInterval</code> is used and <code>timeout</code> is not used, <code>pollInterval</code> will timeout in 60 seconds.	
<code>pollInterval</code>	If this option is not specified, <code>runProcedure</code> returns immediately. If it is specified, <code>runProcedure</code> waits until the job completes. This argument requires setting a value in <i>seconds</i> to determine how often ectool queries the Commander server for job status, but this is not an indefinite activity - set the <code>timeout</code> value to extend the <code>pollInterval</code> for longer than 60 seconds if needed.
<code>timeout</code>	This argument requires a value set in <i>seconds</i> . If <code>pollInterval</code> is specified, this timeout causes <code>runProcedure</code> to stop waiting for the job to complete. It does not stop the job itself. If <code>pollInterval</code> is used and <code>timeout</code> is not used, <code>pollInterval</code> will timeout in 60 seconds.

Positional arguments

`projectName`, `procedureName`, `scheduleName`

Response

The new job ID number.

ec-perl

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

Example

```
$cmdr->runProcedure("Sample Project", {procedureName => "Delay",
  actualParameter => {actualParameterName => "Delay Time", value => 10}});
$xpath = $ec->runProcedure("BSHTest",
  {procedureName => "FakeMotoBuild",
  actualParameter => [
    {actualParameterName => "builddir", value => $cwd},
```



```

    {actualParameterName => "board", value => $board},
    {actualParameterName => "myview", value => $cwv},
    {actualParameterName => "resourcetouse",
      value => $resourcetouse},
  });

```

ectool

syntax: `ectool runProcedure <project name> <procedureName> ...`

Examples

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

```
ectool runProcedure "Sample Project" --procedureName "Delay"
--actualParameter "Delay Time=10"
```

[Back to Top](#)

setJobName

Sets the name of a running job.

You must specify `jobId` and `newName`.

Notes:

The `jobId` can be omitted if the command is run as part of an ElectricCommander step.

A job cannot be renamed after it has completed.

Arguments	Descriptions
<code>jobId</code>	The ID or name of the job you want to rename. The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>newName</code>	Supply any name of your choice to rename the job.

Positional arguments

`jobId`, `newName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->setJobName(<jobId>, <newName>);`

Examples

```
$cmdr->setJobName(4fa765dd-73f1-11e3-b67e-b0a420524153, "Delay Test_541"); (from 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

ElectricCommander includes a powerful built-in scheduler for both managing execution and real-time reporting for a "running" process. Most Commander Installations choose to use its built-in scheduler because it is more powerful than most in-house built and other scheduling solutions.

However, there are use cases where an external scheduler may be appropriate, for example, an LSF Grid engine. Often, such systems are quite mature and may have been in use for many years. An organizations reliance on an LSF Grid system can mandate it remain as the driving scheduler. Many schedulers lack the richness in their graphical user interface, which is an area where Commander excels—especially as it applies to monitoring the status of complex processes and workflows as they progress in real-time through the Commander system. The Commander GUI also provides powerful auditing capabilities for reviewing results of complex process runs.

External Job APIs are designed to leverage the Commander GUI to display results for jobs running on external schedulers. The external scheduler can issue calls through these APIs to provide a representation of this same job within the Commander Jobs page. Commander users and the external scheduler can then monitor the complete integrated system through a single interface—the Commander GUI.

The external system need not be a formal scheduler. In fact, even a simple script might be able to leverage the External Job Step API. For example, a build script could issue API calls at its beginning and end so the build is represented in Commander as a job.

Using the API does NOT consume agent resources. The API simply allows for graphical representation of external jobs within Commander.

completeJob

Completes an externally managed job. Marks the job's root step so the job is marked "completed" when all child steps are completed, and updates the run time for the root step.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>force</code>	<p><Boolean flag - 0 1 true false> If true, all external steps belonging to the job will be marked "complete". Determines whether all external steps under the job should be recursively marked "complete".</p> <p>Note: If this API is called on a job launched with <code>runProcedure</code>, there is no effect unless <code>force</code> is enabled, in which case only external steps are affected.</p>
<code>outcome</code>	<p>Possible values for <code>outcome</code>:</p> <ul style="list-style-type: none"> <code>success</code> - The job finished successfully. <code>warning</code> - The job completed with no errors, but encountered some suspicious conditions. <code>error</code> - The job has finished execution with errors. <p>If specified, the outcome overrides any previously propagated outcome value.</p>

Positional arguments

`jobId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool completeJob <jobId>

Example

```
ectool completeJob 1234
```

[Back to Top](#)

completeJobStep

Completes an externally managed job step. Marks the job step "completed" when all child steps are completed and updates the step run time.

You must specify a `jobStepId`.

Arguments	Descriptions
<code>exitCode</code>	The step's exit code.
<code>force</code>	<p><i><Boolean flag - 0 1 true false></i> If true, all external steps under the job should be recursively marked "complete".</p> <p>Note: If this API is called on a job launched with <code>runProcedure</code>, there is no effect unless <code>force</code> is enabled, in which case only external steps are affected.</p>
<code>jobStepId</code>	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.
<code>outcome</code>	<p>Possible values for <code>outcome</code>:</p> <ul style="list-style-type: none"><code>success</code> - The job step finished successfully.<code>warning</code> - The job step completed with no errors, but encountered some suspicious conditions.<code>error</code> - The job step has finished execution with errors.<code>skipped</code> - The job step was skipped.

Positional arguments

`jobStepId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool completeJobStep <jobStepId>

Example

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

[Back to Top](#)

createJob

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

You must specify `projectName` or `destinationProject`.

Arguments	Descriptions
<code>destinationProject</code>	If specified, determines the project where the job will reside. You must have <i>modify</i> permission on the destination project. <code>projectName</code> or <code>destinationProject</code> must be specified to determine the project where the job is created, <code>destinationProject</code> is preferred.
<code>jobNameTemplate</code>	If specified, the job name will be generated by expanding this argument value. Note: The job name is generated by expanding the <code>jobNameTemplate</code> argument or the <code>jobNameTemplate</code> from the procedure or the system default.
<code>procedureName</code>	If specified, <code>projectName</code> and <code>procedureName</code> are used as a template for the job. You must have <i>execute</i> permission on the procedure. Note: The job name is generated by expanding the <code>jobNameTemplate</code> argument or the <code>jobNameTemplate</code> from the specified procedure or the system default.

Arguments	Descriptions
<code>projectName</code>	The name of the project where this job will reside. You must have <i>modify</i> permission on the destination project. <code>projectName</code> or <code>destinationProject</code> must be specified to determine the project where the job is created. If both are specified, <code>destinationProject</code> is preferred.
<code>status</code>	<code><pending runnable scheduled running></code> The <code>status</code> argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use <code>modifyJob</code> to update the status. Defaults to <code>pending</code> . Possible values for <code>status</code> : <code>pending</code> - The job is not yet runnable. <code>runnable</code> - The job is ready to run. <code>scheduled</code> - The job is scheduled to run. <code>running</code> - The job is executing.

Positional arguments

None

Response

The new job ID number.

ec-perl

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

Example

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

ectool

syntax: `ectool createJob ...`

Example

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

[Back to Top](#)

createJobStep

Use this API to add Commander managed job steps to a running job or job step as well as to create externally managed steps (if "external" is set).

You must specify the parent job step using either the `jobStepId` or `parentPath` arguments (COMMANDER_JOBSTEPID implicitly sets `jobStepId`). The parent job step status must not be `completed`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the <i>called</i> procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. For more information about parameters, click here .
<code>alwaysRun</code>	If set to 1, indicates this job step will run even if the job is aborted before the job step completes. A useful argument for running a "cleanup" step that should run whether the job step is successful or not. The value for <code>alwaysRun</code> is a <i><Boolean flag - 0 1 true false></i> . Defaults to "false".
<code>broadcast</code>	Use this flag to run the same job step on several resources at the same time. The job step is "broadcast" to all resources listed in the <code>resourceName</code> argument. The <code>broadcast</code> value = <i><Boolean flag - 0 1 true false></i> . This argument is applicable only to command job steps. Defaults to "false".
<code>command</code>	The command to run. This argument is applicable to command job steps only.
<code>condition</code>	If empty or non-zero, the job step will run. If set to "0", the job step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the job steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.
<code>credential</code>	Refers to one or more credentials to attach to this job step. These are "dynamic" credentials, captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded. For more information about credentials, see the Credentials and User Impersonation Help topic.

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

Arguments	Descriptions
<code>exclusiveMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>None</code> - the "default", which does not retain a resource. • <code>Job</code> - keeps the resource for the duration of the job step. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a job step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the job steps and its step limit is not exceeded. • <code>Step</code> - keeps the resource for the duration of the job step. • <code>Call</code> - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps.
<code>external</code>	<p>If set, indicates this job step is an external step. Commander will not schedule or run agent commands for external steps, but instead, represents a step managed outside of Commander. The typical usage is with an external Job (see <code>createJob</code>). The status of an external job step is set using <code>modifyJobStep</code>, and it can be completed using <code>completeJobStep</code>. The value for external is a <i><Boolean flag -0 1 true false></i>. Default is "false".</p>
<code>jobStepId</code>	<p>ID of the parent job step. If both <code>jobStepId</code> and <code>parentPath</code> are specified, <code>parentPath</code> is preferred.</p>
<code>jobStepName</code>	<p>The name of the job step. You can use any name of your choice.</p>
<code>logFileName</code>	<p>A custom log file name produced by running the job step. By default, Commander assigns a unique name for this file.</p>
<code>parallel</code>	<p>If set, indicates this job step should run at the same time as adjacent job steps marked to run as parallel also. The value for <code>parallel</code> is a <i><Boolean flag -0 1 true false></i>. Defaults to "false".</p>
<code>parentPath</code>	<p>The path of the parent job step. If both <code>jobStepId</code> and <code>parentPath</code> are specified, <code>parentPath</code> is preferred.</p>

Arguments	Descriptions
<code>postProcessor</code>	The name of a program to run after a job step completes. This program looks at the job step output to find errors and warnings. Commander includes a customizable program called "postp" for this purpose. The value for <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).

Arguments	Descriptions
precondition	<p>The <code>precondition</code> property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, <code>false</code>, or <code>"0"</code>, the step remains in the pending state. Any other value allows the step to proceed to the runnable state.</p> <p>Note: A precondition property allows steps to be created with "pause", which then pauses the procedure. In a paused state, all currently running steps continue, but no additional steps will start.</p> <p>Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (<code>/myJob/buildInstallerCompleted=1</code>). • Set a precondition in step 4: <code>\$/myJob/buildInstallerCompleted</code>
procedureName	The name of the procedure that will contain this job step.

Arguments	Descriptions
projectName	The name of the project that contains the procedure where you are adding a new job step.
releaseExclusive	<p><Boolean flag - 0 1 true false> Declares whether or not this job step will release its resource, which is currently held "exclusively".</p> <p>Note: Setting this flag to "true" is the same as setting <code>releaseMode</code> to <code>release</code>.</p>
releaseMode	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain". • <code>release</code> - releases the resource at the end of this job step. If the resource for the job step was previously acquired with "Retain exclusive" (either by this job step or some preceding job step), the resource exclusivity is canceled at the end of this job step. The resource is released in the normal way so it may be acquired by other jobs. • <code>releasetojob</code> - allows a job step to promote a "step exclusive" resource to a Job exclusive resource.
resourceName	The name of the resource you want this job step to use.
shell	Where <i>shell</i> is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (<code>cmd</code> for Windows, <code>sh</code> for UNIX). Applicable to command steps only.

Arguments	Descriptions
status	<p><pending runnable scheduled running></p> <p>The <code>status</code> argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use <code>modifyJobStep</code> to update the status. Defaults to <code>pending</code>.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"> <code>pending</code> - The job step is not yet runnable. <code>runnable</code> - The job step is ready to run. <code>scheduled</code> - The job step is scheduled to run. <code>running</code> - The job step is executing.
stepName	The name of the new job step you are creating. You can use any name of your choice.
subprocedure	The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.
subproject	If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.
timeLimit	<p>The maximum length of time the job step is allowed to run. After the time specified, the job step will be aborted.</p> <p>The time limit is specified in units that can be hours, minutes, or seconds.</p>
timeLimitUnits	Specify <code>hours minutes seconds</code> for time limit units.
workingDirectory	<p>The Commander agent sets this directory as the "current working directory," when running the command contained in the job step. If no working directory is specified in the job step, Commander uses the directory it created for the job in the Commander workspace as the working directory.</p> <p>Note: If running a job step on a proxy resource, this directory must exist on the proxy target.</p>
workspaceName	The name of the workspace where this job step's log files will be stored.

Positional arguments

jobStepId or parentPath

Response

Returns a `jobStep` object.

ec-perl

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

Examples

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

```
$cmdr->createJobStep ({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153", external => "1"});
```

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

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

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

```
$cmdr->createJobStep(  
  {  
    projectName => 'Default',  
    subprocedure => 'coolProcedure',  
    actualParameter => [  
      {  
        actualParameterName => 'sshCredentialParameter',  
        value => 'sshCredentialParameter'  
      }  
    ],  
    credential => [  
      {  
        credentialName => 'sshCredentialParameter',  
        userName => 'sshUser',  
        password => 'super_secure_sshPassword'  
      }  
    ]  
  }  
);
```

```

# Create two parallel job steps and send them to the Commander server using the batch API.

# Create the batch API object
my $batch = $ec->newBatch('parallel');

# Create multiple requests
my @reqIds = (
    $batch->createJobStep(
        {
            parallel      => '1',
            projectName   => 'Default',
            subprocedure   => 'coolProcedure',
            actualParameter => [
                {
                    actualParameterName => 'input',
                    value                => 'helloWorld'
                }
            ],
        }
    ),
    $batch->createJobStep(
        {
            parallel      => '1',
            projectName   => 'Default',
            subprocedure   => 'coolProcedure',
            actualParameter => [
                {
                    actualParameterName => 'input',
                    value                => 'helloWorld'
                }
            ],
        }
    ),
);

# Send off the requests

```

```
$batch->submit();
```

ectool

syntax: ectool createJobStep ...

Examples

```
ectool createJobStep --parentPath /jobs/123 --external 1
```

```
ectool createJobStep --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153 --external 1
```

```
ectool createJobStep --parallel 1 --projectName Default --subprocedure
```

```
coolProcedure --actualParameter input=helloWorld
```

[Back to Top](#)

modifyJob

Modifies the status of an externally managed job.

You must specify a `jobId`.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>status</code>	<p><pending runnable scheduled running></p> <p>The <code>status</code> argument determines the current status of the job, and also sets related timing values.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"><code>pending</code> - The job is not yet runnable.<code>runnable</code> - The job is ready to run.<code>scheduled</code> - The job is scheduled to run.<code>running</code> - The job is executing.

Positional arguments

`jobId`

Response

The `jobId` element.

ec-perl

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

Example

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

ectool

syntax: ectool modifyJob <jobId> ...

Example

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

[Back to Top](#)

modifyJobStep

Modifies the status of an externally managed job step.

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

Arguments	Descriptions
jobStepId	The Commander-generated ID for the job step.
status	<p><pending runnable scheduled running> The <code>status</code> argument determines the current status of the job, and also sets related timing values.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"> pending - The job step is not yet runnable. runnable - The job step is ready to run. scheduled - The job step is scheduled to run. running - The job step is executing.

Positional arguments

jobStepId

Response

Returns a modified `jobStep` object.

ec-perl

syntax: \$cmdr->modifyJobStep (<jobStepId>, {<optional>});

Example

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

ectool

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

Example

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

[Back to Top](#)

waitForJob

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

Arguments	Descriptions
<code>jobId</code>	The job to wait for. The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>finalStatus</code>	The status to wait for. Must be either "running" or "completed" (default is "completed").
<code>timeout</code>	The number of seconds to wait before giving up on a request.

Positional arguments

`jobId`

Response

Returns the result from the final `getJobStatus` query.

ec-perl

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

Example

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

[Back to Top](#)

API Commands - Parameter Management

`attachParameter`
`createActualParameter`
`createFormalParameter`
`deleteActualParameter`
`deleteFormalParameter`
`detachParameter`
`getActualParameter`
`getActualParameters`
`getFormalParameter`
`getFormalParameters`
`modifyActualParameter`
`modifyFormalParameter`

attachParameter

Attaches a formal parameter to a step.

Attaching a parameter allows a step to use the credential (passed in a parameter) as an actual parameter to a subprocedure

call or directly in a `getFullCredential` call in a command step. For more information about parameters, click [here](#).

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

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

Positional arguments

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

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->attachCredential("Test Proj", "Run Build", "Get Sources", "SCM Credential1");
```

ectool

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

Example

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

[Back to Top](#)

createActualParameter

Creates a new actual parameter for a step that calls a nested procedure. The parameter is passed to the nested procedure when the step runs. At run time, the actual parameter name must match the name of a formal parameter in the nested procedure.

Passing Actual Parameters

You can use actual parameters in three types of API calls:

- calling `runProcedure` to start a new job
- setting up a schedule
- creating or modifying a subprocedure step

For example, when you call `runProcedure` using `ectool`, set the actual parameters to the procedure on the command line using the optional argument `--actualParameter`, followed by a list of *name/value* pairs. The following is an example of calling a procedure named `MasterBuild`:

```
ectool runProcedure "project A" --procedureName "MasterBuild"
--actualParameter Branch=main Type=Debug
```

To make this call using the Perl API, define a list. Each element of the list is an anonymous hash reference that specifies one of the actual parameters. Now you can pass a reference to the list as the value of the `actualParameter` argument.

Here is the same example called via the Perl API:

```
# Run the procedure
$xPath = $cmdr->runProcedure("project A",
    {procedureName => "MasterBuild",
      actualParameter => [
        {actualParameterName => 'Branch',
          value => 'main'},
        {actualParameterName => 'Type',
```

```

        value => 'Debug'},
    ]});

```

Specifying most arguments to the `createStep` API in Perl is fairly intuitive; like any other API, you specify key-value pairs in a hash argument for all optional parameters. However, specifying actual parameters is more involved because actual parameters are not arbitrary key-values characterizing the step. Instead, they are key-values characterizing actual parameters to the step. See the following `createStep` request in XML:

```

<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
    <value>val2</value>
  </actualParameter>
</createStep>

```

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

```

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

```

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

```

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

```

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

Arguments	Descriptions
<code>actualParameterName</code>	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.
<code>procedureName</code>	The name of the procedure containing the step.
<code>projectName</code>	The name of the project containing the procedure.

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

Positional arguments

projectName, procedureName, stepName, actualParameterName

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->createActualParameter("Sample Project", "CallSub", "Step1", "Extra Parm",  
    {value => "abcd efg"});
```

ectool

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

Example

```
ectool createActualParameter "Sample Project" "CallSub" "Step1" "Extra Parm"  
    --value "abcd efg"
```

[Back to Top](#)

createFormalParameter

Creates a new formal parameter.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
defaultValue	This value is used for the formal parameter if a value is not supplied by the caller.
expansionDeferred	<i><Boolean flag - 0 1 true false></i> Default is "false," which means the formal parameter is expanded immediately.
formalParameterName	The name for this parameter - used when the procedure is invoked to specify a value for the parameter.
procedureName	The name of the procedure containing the parameter. Note: In releases earlier than ElectricCommander 5.0, <code>procedureName</code> is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, <code>procedureName</code> is optional.
projectName	The name of the project containing the procedure.
required	<i><Boolean flag - 0 1 true false></i> If set to 1, this value indicates whether a non-blank value must be supplied when calling the procedure.
stateDefinitionName	The name of the state definition.
type	The "type" can be any string value. Used primarily by the web interface to represent custom form elements. However, if "credential" is the string value, the server will expect a credential as the parameter value.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, for procedure parameters:
`projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, for procedure parameters: `projectName`, `procedureName`, and `formalParameterName`.

For workflow state parameters: `projectName`, `formalParameterName`, `workflowDefinitionName` and `stateDefinitionName`

Response

None or status OK message.

ec-perl

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

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

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

Example

```
$cmdr->createFormalParameter("Sample Project", "Branch Name", {required => 1 });
```

Examples using parameters to create a check box, radio button, and drop-down box

Check box example:

```
$ec->createFormalParameter(
    $newProjectName,
    "$buildprocedurename",
    'CheckoutSources',
    {
        type => "checkbox",
        required => 0,
        defaultValue => 'true',
        description => "If checked, update the sandbox from Subversion (turn
            off for debugging only)."
    }
);
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
    ec_customEditorData/parameters/CheckoutSources/checkedValue", "true");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
    ec_customEditorData/parameters/CheckoutSources/uncheckedValue", "false");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
    ec_customEditorData/parameters/CheckoutSources/initiallyChecked", "0");
```

Radio button example:

```
$ec->createFormalParameter(
    $newProjectName,
    "$buildprocedurename",
    'BuildType',
    {
        type => "radio",
        required => 1,
        defaultValue => '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");
```

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/select", "list");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
  ec_customEditorData/parameters/BuildType/options/option1/text", "one");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
  ec_customEditorData/parameters/BuildType/options/option1/value", "1");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
  ec_customEditorData/parameters/BuildType/options/option2/text", "two");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/
  ec_customEditorData/parameters/BuildType/options/option2/value", "2");
```

ectool

For procedure parameters

syntax: ectool createFormalParameter <projectName> <formalParameterName> ...

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

syntax: ectool createFormalParameter <projectName> <procedureName>
<formalParameterName> ...

Example

```
ectool createFormalParameter "Sample Project" "Branch Name" --required 1
```

For workflow state parameters

syntax: ectool createFormalParameter --formalParameterName <name>
--projectName <name> --workflowDefinitionName <name> --stateDefinitionName <name>

Example using parameters to create a check box

You must create the `ec_customEditorData` property to add other parameters to the check box.

```
ectool createFormalParameter --projectName $projectName --workflowDefinitionName $workflowDefinitionName
--stateDefinitionName $stateName --formalParameterName $parameter_name --type "checkbox" --defaultValue "true"

#Create properties for the check box (required):#Create properties for the check box (required)
ectool createProperty ec_customEditorData --projectName $projectName --workflowDefinitionName $workflowDefinitionName
--stateDefinitionName $stateName --propertyType sheet

ectool --silent setProperty "/projects/$projectName/workflowDefinitions/$workflowDefinitionName
/stateDefinitions/$stateName/properties/ec_customEditorData/parameters/$parameter_name/checkedValue/" "true"

ectool setProperty "/projects/$projectName/workflowDefinitions/$workflowDefinitionName
/stateDefinitions/$stateName/properties/ec_customEditorData/parameters/$parameter_name/initiallyChecked/" "1"

ectool setProperty "/projects/$projectName/workflowDefinitions/$workflowDefinitionName
/stateDefinitions/$stateName/properties/ec_customEditorData/parameters/$parameter_name/uncheckedValue/" "false"
```

[Back to Top](#)

deleteActualParameter

Deletes an actual parameter.

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

Arguments	Descriptions
<code>actualParameterName</code>	The name of the actual parameter you want to delete.
<code>procedureName</code>	The name of the procedure that contains the step with this parameter.
<code>projectName</code>	The name of the project that contains this actual parameter.
<code>scheduleName</code>	The name of the schedule containing the actual parameter.
<code>stateDefinitionName</code>	The name of the state definition.

Arguments	Descriptions
stepName	The name of the step that contains this actual parameter you want to delete.
transitionDefinitionName	The name of the transition definition.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, procedureName, stepName, actualParameterName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>);

Example

```
$cmdr->deleteActualParameter('Sample Project', 'CallSub', 'Step1', 'Different Parm');
```

ectool

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

Example

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

[Back to Top](#)

deleteFormalParameter

Deletes a formal parameter.

You must specify projectName and formalParameterName.

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

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure that contains this parameter. Note: In releases earlier than ElectricCommander 5.0, <code>procedureName</code> is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, <code>procedureName</code> is optional.
<code>projectName</code>	The name of the project that contains the procedure/parameter you want to delete.
<code>stateDefinitionName</code>	The name of the state definition.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, `projectName`, `procedureName`, and `formalParameterName`.

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteFormalParameter(<projectName>, <formalParameterName>);`

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

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

Example

```
$cmdr->deleteFormalParameter("Sample Project", "Build Name");
```

ectool

syntax: `ectool deleteFormalParameter <projectName> <formalParameterName>`

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

syntax: `ectool deleteFormalParameter <projectName> <procedureName> <formalParameterName>`

Example

```
ectool deleteFormalParameter "Sample Project" "Build Name"
```

[Back to Top](#)

detachParameter

Detaches a formal parameter from a step.

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

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

Positional arguments

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

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

getActualParameter

Retrieves an actual parameter by its name. For more information about parameters, click [here](#).

You must specify an `actualParameterName`. If you need actual parameters on a step, the following 3 arguments must be used together to specify a step: `projectName`, `procedureName`, and `stepName`.

Arguments	Descriptions
<code>actualParameterName</code>	The name of the actual parameter.
<code>applicationName</code>	The name of the application, if the actual parameter is on an application process step; must be unique among all projects.
<code>componentName</code>	The name of the component, if the actual parameter is on a component process step.
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created. Supply this argument to query a subprocedure call to the job step's parameter.
<code>procedureName</code>	The name of the procedure to query for the procedure step's parameter.
<code>processName</code>	The name of the process, if the actual parameter is on a process step.
<code>processStepName</code>	The name of the process step, if the actual parameter is on a process step.
<code>projectName</code>	The name of the project to query for a schedule or procedure step's parameter.
<code>scheduleName</code>	The name of the schedule to query for the schedule's parameter.
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step to query for the step's parameter.
<code>transitionDefinitionName</code>	The name of the transition definition.

Arguments	Descriptions
transitionName	The name of the transition.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

actualParameterName

Response

One [actualParameter](#) element.

ec-perl

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

Example

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

ectool

syntax: ectool getActualParameter <actualParameterName> ...

Example

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

[Back to Top](#)

getActualParameters

Retrieves all actual parameters from a job, job step, schedule, or step. For more information about parameters, click [here](#).

You must specify object locators to find the parameter. If finding parameters on a step, you must use `projectName`, `procedureName`, and `stepName` to specify a step.

Arguments	Descriptions
applicationName	The name of the application, if the actual parameter is on an application process step; must be unique among all projects.

Arguments	Descriptions
<code>componentName</code>	The name of the component, if the actual parameter is on a component process step.
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created.
<code>projectName</code>	The name of the project containing these parameters.
<code>procedureName</code>	The name of the procedure containing these parameters.
<code>processName</code>	The name of the process, if the actual parameter is on a process step.
<code>processStepName</code>	The name of the process step, if the actual parameter is on a process step.
<code>scheduleName</code>	The name of the schedule containing parameters.
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step containing parameters.
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.

Positional arguments

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

Response

Zero or more `actualParameter` elements.

ec-perl

syntax: \$cmdr->getActualParameters{{...}};

Example

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

ectool

syntax: ectool getActualParameters ...

Example

```
ectool getActualParameters --projectName "Sample Project"
    --procedureName "CallSub" --stepName "Step1"
```

[Back to Top](#)

getFormalParameter

Retrieves a formal parameter by its name.

You must specify `projectName` and `formalParameterName`.

Arguments	Descriptions
<code>formalParameterName</code>	The name of the formal parameter.
<code>procedureName</code>	The name of the procedure containing the formal parameter. Note: In releases earlier than ElectricCommander 5.0, <code>procedureName</code> is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, <code>procedureName</code> is optional.
<code>projectName</code>	The name of the project containing the procedure.
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.

Positional arguments

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, `projectName`, `procedureName`, and `formalParameterName`.

Response

One `formalParameter` element.

ec-perl

syntax: `$cmdr->getFormalParameter(<projectName>, <formalParameterName>);`

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

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

Example

```
$cmdr->getFormalParameter("Test", "Get Sources");
```

ectool

syntax: `ectool getFormalParameter<projectName> <formalParameterName>`

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

syntax: `ectool getFormalParameter<projectName> <procedureName> <formalParameterName>`

Example

```
ectool getFormalParameter Test "Get Sources"
```

[Back to Top](#)

getFormalParameters

Retrieves all formal parameters from a procedure, schedule, or step.

You must specify locator arguments to identify a procedure, schedule, or subprocedure step. If the locators identify a schedule or step, the formal parameters of the called procedure are returned.

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure. Also requires the <code>projectName</code>
<code>projectName</code>	The name of the project containing the object whose parameters are being retrieved.

Arguments	Descriptions
<code>scheduleName</code>	The name of the schedule. Also requires the <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step. Also requires the <code>projectName</code> and <code>procedureName</code>
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.

Positional arguments

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

Response

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

ec-perl

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

Example

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

ectool

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

Example

```
getFormalParameters Test --procedureName Build
```

[Back to Top](#)

modifyActualParameter

Modifies an existing actual parameter. An actual parameter is a name/value pair passed to a subprocedure. This command supports renaming the actual parameter and setting its value. For more information about parameters, click [here](#).

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

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

modifyFormalParameter

Modifies an existing formal parameter.

Arguments	Descriptions
defaultValue	This value is used for the formal parameter if one is not supplied by the caller.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
expansionDeferred	<Boolean flag - 0 1 true false> Default is "false," which means the formal parameter is expanded immediately.
formalParameterName	The name for this formal parameter. Used when the procedure is invoked to specify a value for the parameter.
newName	Supply any name of your choice to rename the parameter.
procedureName	The name of the procedure containing this parameter. Note: In releases earlier than ElectricCommander 5.0, <code>procedureName</code> is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, <code>procedureName</code> is optional.
projectName	The name of the project containing this parameter.
required	<Boolean flag - 0 1 true false> If set to 1, this value indicates whether a non-blank value must be supplied when calling the procedure.
stateDefinitionName	The name of the state definition.
type	<code>type</code> can be any string value. Used primarily by the web interface to represent custom form elements. However, if "credential" is the string value, the server will expect a credential as the parameter value.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, for procedure parameters:

`projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, for procedure parameters: `projectName`,

`procedureName`, and `formalParameterName`.

For workflow state parameters: `projectName`, `formalParameterName`, `workflowDefinitionName` and `stateDefinitionName`

Response

None or a status OK message.

ec-perl

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

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

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

Example

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

ectool

For procedure parameters:

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

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

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

Example

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

For workflow state parameters:

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

[Back to Top](#)

API Commands - Plugin Management

`deletePlugin`
`getPlugin`
`getPlugins`
`installPlugin`
`modifyPlugin`
`promotePlugin`
`uninstallPlugin`

deletePlugin

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

You must specify a `pluginName`.

Arguments	Descriptions
<code>pluginName</code>	The name of the plugin you want to delete.

Positional arguments

`pluginName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deletePlugin(<pluginName>);`

Example

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

ectool

syntax: `ectool deletePlugin <pluginName>`

Example

```
ectool deletePlugin TheWidget-1.0
```

[Back to Top](#)

getPlugin

Retrieves an installed plugin.

You must specify the `pluginName`.

Arguments	Descriptions
<code>pluginName</code>	The name of the plugin to find. If the name is specified without a version number, the currently promoted version is returned if possible.

Positional arguments

`pluginName`

Response

One `plugin` element, which includes the plugin ID, name, time created, label, owner, key, version, and more.

ec-perl

syntax: `$cmdr->getPlugin(<pluginName>);`

Example

```
$cmdr->getPlugin("TheWidget");
```

ectool

syntax: `ectool getPlugin <pluginName>`

Example

```
ectool getPlugin TheWidget
```

[Back to Top](#)

getPlugins

Retrieves all installed plugins.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `plugin` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getPlugins`

Example

```
ectool getPlugins
```

[Back to Top](#)

installPlugin

Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.

You must specify the `url`.

Arguments	Descriptions
<code>force</code>	<i><Boolean flag - 0 1 true false></i> Specifying false causes an existing plugin with the same key and version to be overwritten with the new plugin contents, otherwise an error is returned.
<code>url</code>	The location of the plugin JAR file to install. If the location refers to a file on the client machine, the file will be uploaded to the server. If the location refers to a remote accessible file (for example, via an <code>http://url</code>), the server will download it. If the location is a <code>file:</code> reference, the file will be read directly from the specified location on the server's file system.

Positional arguments

`url`

Response

One `plugin` element.

ec-perl

syntax: `$cmdr->installPlugin(<url>, {...});`

Example

```
$cmdr->installPlugin("./myPlugin.jar")
```

ectool

syntax: `ectool installPlugin <url> ...`

Example

```
ectool installPlugin ./myPlugin.jar
```

[Back to Top](#)

modifyPlugin

Modifies an existing plugin.

Note: Some plugin attributes available on the Plugins web page are not available in any of the plugin-related APIs.

Because some plugin meta data comes from the `plugin.xml` file, the web server can access this data, but the Commander

server cannot. Thus, the Plugin Manager, run in the web server context, provides additional information and functionality.

You must specify the `pluginName`.

Arguments	Descriptions
<code>author</code>	The author of the plugin.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>label</code>	The name of the plugin as displayed on the Plugins web page.
<code>pluginName</code>	The name of the plugin to modify. If the name is specified without a version number, the currently promoted version is used if possible.

Positional arguments

`pluginName`

Response

One [plugin](#) element.

ec-perl

syntax: `$cmdr->modifyPlugin(<pluginName>, {...});`

Example

```
$cmdr->modifyPlugin('TheWidget', {description => "new description"});
```

ectool

syntax: ectool modifyPlugin <pluginName> ...

Example

```
ectool modifyPlugin TheWidget --description "new description"
```

[Back to Top](#)

promotePlugin

Sets the promoted flag on a plugin. Only one version of a plugin can be promoted at a time, so setting the promoted flag to "true" on one version sets the flag to false on all other plugins with the same key. The promoted version is the one resolved by an indirect reference of the form `$/plugins/<key>` or a plugin name argument without a specified version.

You must specify the `pluginName`.

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

Positional arguments

`pluginName`

Response

One `plugin` element, which includes the plugin ID, name, time created, label, owner, key, version, project name, and more.

ec-perl

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

Example

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

ectool

syntax: `ectool promotePlugin <pluginName> ...`

Example

```
ectool promotePlugin TheWidget-1.0
```

[Back to Top](#)

uninstallPlugin

Uninstalls a plugin, deleting the associated project and any installed files.

You must specify the `pluginName`.

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

Positional arguments

`pluginName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool uninstallPlugin <pluginName> ...`

Example

```
ectool uninstallPlugin TheWidget-1.0
```

[Back to Top](#)

API Commands - Procedure Management

[createProcedure](#)
[createStep](#)
[deleteProcedure](#)
[deleteStep](#)
[getProcedure](#)
[getProcedures](#)
[getStep](#)
[getSteps](#)
[modifyProcedure](#)
[modifyStep](#)
[moveStep](#)

createProcedure

Creates a new procedure for an existing project.

You must specify `projectName` and `procedureName`.

Arguments	Descriptions
<code>credentialName</code>	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<i>cred1</i>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>description</code>	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
<code>jobNameTemplate</code>	Template used to determine the default name of jobs launched from a procedure.
<code>procedureName</code>	The name you define for this procedure. You can use any name of your choice.

Arguments	Descriptions
<code>projectName</code>	The name of the project that contains this procedure.
<code>resourceName</code>	The name of a resource or pool to use as the default for steps run by this procedure.
<code>timeLimit</code>	If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from <code>runProcedure</code> (or a schedule), the time limit acts as a global job timeout. The "timer" for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).
<code>timeLimitUnits</code>	Time limit units are <code>hours minutes seconds</code>
<code>workspaceName</code>	The name of the workspace to use as the default for steps run by this procedure.

Positional arguments

`projectName`, `procedureName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->createProcedure("Test Proj", "Run Build", {resourceName => "Test Resource"});
```

ectool

syntax: `ectool createProcedure <projectName> <procedureName> ...`

Example

```
ectool createProcedure "Test Proj" "Run Build" --resourceName "Test Resource"
```

[Back to Top](#)

createStep

Use this command to create a new procedure step.

Fundamentally, ElectricCommander supports three types of steps:

- Command Step - the step executes a command or script under the control of a shell program.
- Subprocedure Step - the step invokes another Commander procedure. In this case, the step will not complete until all subprocedure steps have completed.
- Custom Step

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

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. For more information about parameters, click here .
<code>alwaysRun</code>	If set to 1, indicates this step will run even if the job is aborted before the step completes. A useful argument for running a "cleanup" step that should run whether the job is successful or not. The value for <code>alwaysRun</code> is a <i><Boolean flag -0 1 true false></i> . Defaults to "false".
<code>broadcast</code>	Use this flag to run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the <code>resourceName</code> argument. The <code>broadcast</code> value = <i><Boolean flag -0 1 true false></i> . This argument is applicable only to command steps. Defaults to "false".
<code>command</code>	The command to run. This argument is applicable to command steps only.
<code>commandFile</code>	This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. Contents of the <i>command file</i> is read and stored in the "command" field. This is an alternative argument for <code>command</code> and is useful if the "command" field spans multiple lines. The <code>commandFile</code> value is the actual <i>command file</i> text. This argument is applicable to command steps only.

Arguments	Descriptions
condition	If empty or non-zero, the step will run. If set to "0", the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.
credentialName	The credential to use for impersonation on the agent. credentialName can be one of two forms: relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
errorHandling	Determines what happens to the procedure if the step fails: <ul style="list-style-type: none"> failProcedure - The current procedure continues, but the overall status is error (default). abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete. abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure. abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run. abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps. ignore - Continues as if the step succeeded.
exclusive	If set to 1, indicates this step should acquire and retain this resource exclusively. The value for exclusive is a <i>Boolean flag</i> <code>-0 1 true false></code> . Defaults to "false". Note: Setting exclusive, sets exclusiveMode to "job".

Arguments	Descriptions
<code>exclusiveMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>None</code> - the "default", which does not retain a resource. • <code>Job</code> - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the steps and its step limit is not exceeded. • <code>Step</code> - keeps the resource for the duration of the step. • <code>Call</code> - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.
<code>logFileName</code>	A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.
<code>parallel</code>	If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. The value for <code>parallel</code> is a <i><Boolean flag -0 1 true false></i> . Defaults to "false".
<code>postProcessor</code>	The name of a program to run after a step completes. This program looks at the step output to find errors and warnings. Commander includes a customizable program called "postp" for this purpose. The value for <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).

Arguments	Descriptions
precondition	<p>By default, if the step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated. A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (/myJob/buildInstallerCompleted=1). • Set a precondition in step 4: \$[/myJob/buildInstallerCompleted]
procedureName	The name of the procedure that will contain this step.
projectName	The name of the project that contains the procedure where you are adding a new step.
releaseExclusive	<p><Boolean flag - 0 1 true false> Declares whether or not this step will release its resource, which is currently held exclusively.</p> <p>Note: Setting this flag to "true" is the same as setting releaseMode to release.</p>

Arguments	Descriptions
<code>releaseMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain". • <code>release</code> - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs. • <code>releasetojob</code> - allows a step to promote a "step exclusive" resource to a Job exclusive resource.
<code>resourceName</code>	The name of the resource you want this step to use.
<code>shell</code>	Where <i>shell</i> is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (<code>cmd</code> for Windows, <code>sh</code> for UNIX). This is applicable to command steps only.
<code>stepName</code>	The name of the new step you are creating. You can use any name of your choice.
<code>subprocedure</code>	The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.
<code>subproject</code>	If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.
<code>timeLimit</code>	<p>The maximum length of time the step is allowed to run. After the time specified, the step will be aborted.</p> <p>The time limit is specified in units that can be hours, minutes, or seconds.</p>
<code>timeLimitUnits</code>	Specify <code>hours minutes seconds</code> for time limit units.

Arguments	Descriptions
<code>workingDirectory</code>	The Commander agent sets this directory as the “current working directory,” when running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace as the working directory. Note: If running a step on a proxy resource, this directory must exist on the proxy target.
<code>workspaceName</code>	The name of the workspace where this step's log files will be stored.

Positional arguments

`projectName`, `procedureName`, `stepName`

Response

None or a status OK message.

ec-perl

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

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

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

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

```
<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
    <value>val2</value>
  </actualParameter>
</createStep>
```

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

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

In other words, the value of the `actualParameter` key in the optional arguments hash is a list of hashes, each

representing one actual parameter. If the subprocedure call only takes one actual parameter, the value of the

`actualParameter` key can be specified as just the hash representing the one parameter:

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

Example

```
$cmdr->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Delay",
actualParameter => {actualParameterName => 'Delay Time', value => '5'}});
```

ectool

syntax: `ectool createStep <projectName> <procedureName> <stepName> ...`

Specifying actual parameters in an `ectool` call is also different than specifying other arguments.

Specify each key-value as an equal-sign delimited value:

```
ectool createStep ... --actualParameter "Delay Time=5" "parm2=val2"
```

Note: If the parameter name or value contains spaces, quotes are needed.

Examples

```
ectool createStep "Test Proj" "Run Build" "Compile" --command "make all"
```

```
ectool createStep "Test Proj" "Run Build" "Common Cleanup" --subprocedure "Delay"
--actualParameter "Delay Time=5"
```

[Back to Top](#)

deleteProcedure

Deletes a procedure, including all steps.

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

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure you want to delete.
<code>projectName</code>	The name of the project that contains this procedure.

Positional arguments

projectName, procedureName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteProcedure(<projectName>, <procedureName>);

Example

```
$cmdr->deleteProcedure("Test Proj", "Run Build");
```

ectool

syntax: ectool deleteProcedure <projectName> <procedureName>

Example

```
ectool deleteProcedure "Test Proj" "Run Build"
```

[Back to Top](#)

deleteStep

Deletes a step from a procedure.

You must specify projectName, procedureName, and stepName.

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

Positional arguments

projectName, procedureName, stepName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteStep(<projectName>, <procedureName>, <stepName>);

Example

```
$cmdr->deleteStep("Test Proj", "Run Build", "Compile");
```

ectool

syntax: `ectool deleteStep <projectName> <procedureName> <stepName>`

Example

```
ectool deleteStep "Test Proj" "Run Build" "Compile"
```

[Back to Top](#)

getProcedure

Finds a procedure by its name.

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

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure you are retrieving.
<code>projectName</code>	The name of the project containing the procedure to retrieve.

Positional arguments

`projectName, procedureName`

Response

One [procedure](#) element, which includes the procedure ID, name, time created, job name template, owner, resource name, workspace name, project name, and more.

ec-perl

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

Example

```
$cmdr->getProcedure("Test Proj", "Run Build");
```

ectool

syntax: `ectool getProcedure <projectName> <procedureName>`

Example

```
ectool getProcedure "Test Proj" "Run Build"
```

[Back to Top](#)

getProcedures

Retrieves all procedures in one project.

You must specify the `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the procedures to retrieve.

Positional arguments

`projectName`

Response

One or more [procedure](#) elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getProcedures <projectName>`

Example

```
ectool getProcedures "Test Proj"
```

[Back to Top](#)

getStep

Retrieves a step from a procedure.

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

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

Positional arguments

`projectName`, `procedureName`, `stepName`

Response

One [step](#) element.

ec-perl

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

Example

```
$cmdr->getStep("Test Proj", "Run Build", "Compile");
```

ectool

syntax: ectool getStep <projectName> <procedureName> <stepName>

Example

```
ectool getStep "Test Proj" "Run Build" "Compile"
```

[Back to Top](#)

getSteps

Retrieves all steps in a procedure.

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

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

Positional arguments

`projectName`, `procedureName`

Response

Zero or more [step](#) elements.

ec-perl

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

Example

```
$cmdr->getSteps("Test Proj", "Run Build");
```

ectool

syntax: ectool getSteps <projectName> <procedureName>

Example

```
ectool getSteps "Test Proj" "Run Build"
```

[Back to Top](#)

modifyProcedure

Modifies an existing procedure.

You must specify `projectName` and `procedureName`.

Arguments	Descriptions
<code>credentialName</code>	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<i>cred1</i>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>description</code>	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
<code>jobNameTemplate</code>	Job name format for jobs created by running this procedure.
<code>newName</code>	Supply any name of your choice to rename the procedure.
<code>procedureName</code>	The name of the procedure to modify.
<code>projectName</code>	<p>The name of the project to modify.</p> <p>Also requires <code>procedureName</code></p>
<code>resourceName</code>	The name of the default resource where steps belonging to this procedure will run. This name may be a resource pool name.
<code>timeLimit</code>	<p>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.</p> <p>The "timer" for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).</p>
<code>timeLimitUnits</code>	Time limit units are <code>hours minutes seconds</code>
<code>workspaceName</code>	The name of the default workspace where job output is stored.

Positional arguments

projectName, procedureName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

modifyStep

Modifies an existing step.

You must specify projectName, procedureName, and stepName.

Arguments	Descriptions
actualParameter	Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure.
alwaysRun	<Boolean flag - 0 1 true false> If set to 1, indicates this step will run even if the job is aborted before the step completes. A useful argument for running a "cleanup" step that should run whether the job is successful or not.
broadcast	<Boolean flag - 0 1 true false> Use this flag to run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the resourceName.

Arguments	Descriptions
<code>clearActualParameters</code>	<Boolean flag - 0 1 true false> If set to true, all actual parameters will be removed from the step.
<code>command</code>	The step command.
<code>commandFile</code>	This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the <i>command file</i> is read and stored in the "command" field. This is an alternative argument for <code>command</code> and is useful if the "command" field spans multiple lines.
<code>condition</code>	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.
<code>credentialName</code>	<code>credentialName</code> can be one of two forms: relative (for example, " <i>cred1</i> ") - the credential is assumed to be in the project that contains the request target object. absolute (for example, " <i>/projects/BuildProject/credentials/cred1</i> ") - the credential can be from any specified project, regardless of the target object's project.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>

Arguments	Descriptions
errorHandling	<p>Determines what happens to the procedure if the step fails:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <ul style="list-style-type: none"> <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. <code>ignore</code> - Continues as if the step succeeded.
exclusive	<p>If set to 1, indicates this step should acquire and retain this resource exclusively. The value for <code>exclusive</code> is a <i><Boolean flag -0 1 true false></i>. Defaults to "false".</p> <p>Note: Setting <code>exclusive</code>, sets <code>exclusiveMode</code> to "job".</p>
exclusiveMode	<p>Use one of the following options:</p> <ul style="list-style-type: none"> <code>None</code> - the "default", which does not retain a resource. <code>Job</code> - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the steps and its step limit is not exceeded. <code>Step</code> - keeps the resource for the duration of the step. <code>Call</code> - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.
logFileName	A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name to this file.
newName	Supply any name of your choice to rename the step.
parallel	<i><Boolean flag - 0 1 true false></i> Indicates if this step should run at the same time as adjacent steps marked to run as parallel also.

Arguments	Descriptions
precondition	<p>By default, if the step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated. A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (/myJob/buildInstallerCompleted=1). • Set a precondition in step 4: \$[/myJob/buildInstallerCompleted]
procedureName	<p>The name of the procedure containing the step to modify. Also requires projectName</p>
projectName	<p>The name of the project containing the step to modify. Also requires procedureName</p>
postProcessor	<p>The name of a program to run (script) after a step completes. This program looks at the step output to find errors and warnings. ElectricCommander includes a customizable program called "postp" for this purpose.</p>

Arguments	Descriptions
<code>releaseExclusive</code>	<p><Boolean flag - 0 1 true false> Declares whether or not this step will release its resource, which is currently held exclusively.</p> <p>Note: Setting this flag to "true" is the same as setting <code>releaseMode</code> to "release".</p>
<code>releaseMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain". • <code>release</code> - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs. • <code>releasetojob</code> - allows a step to promote a Step exclusive resource to a Job exclusive resource.
<code>resourceName</code>	The name of the resource used by this step.
<code>shell</code>	<p>Where <i>shell</i> is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line.</p> <p>Normally, this file is a command shell, but it could be any other command line program. The default is to use the standard shell for the platform it runs on.</p>
<code>stepName</code>	<p>The name of the step.</p> <p>Also requires <code>projectName</code> and <code>procedureName</code></p>
<code>subprocedure</code>	The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandField</code> .
<code>subproject</code>	<p>If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found.</p> <p>By default, the current project is used.</p>
<code>timeLimit</code>	The maximum length of time the step is allowed to run. After the time specified, the step will be aborted.
<code>timeLimitUnits</code>	<hours minutes seconds>

Arguments	Descriptions
<code>workingDirectory</code>	The Commander agent sets this directory as the "current working directory," running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace.
<code>workspaceName</code>	The name of the workspace used by this step.

Positional arguments

`projectName`, `procedureName`, `stepName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyStep <projectName> <procedureName> <stepName> ...`

Example

```
ectool modifyStep "Test Proj" "Run Build" "Compile" --commandFile tempfile.txt
```

[Back to Top](#)

moveStep

Moves a step within a procedure.

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

Arguments	Descriptions
<code>beforeStep</code>	Moves the step (<code>stepName</code>) to position before the step "named" by this option. If omitted, <code>stepName</code> is moved to the end of the list of steps.
<code>procedureName</code>	The name of the procedure containing the step to move.

Arguments	Descriptions
projectName	The name of the project containing the step to move. Also requires procedureName
stepName	The name of the step to move. Also requires projectName and procedureName

Positional arguments

projectName, procedureName, stepName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->moveStep(<projectName>, <procedureName>, <stepName>, {<optionals>});

Example

```
$cmdr->moveStep("Test Proj", "Run Build", "Get Sources", {beforeStep => "Compile"});
```

ectool

syntax: ectool moveStep <projectName> <procedureName> <stepName> ...

Example

```
ectool moveStep "Test Proj" "Run Build" "Get Sources"  
--beforeStep "Compile"
```

[Back to Top](#)

API Commands - Process

[createProcess](#)

[deleteProcess](#)

[getProcess](#)

[getProcesses](#)

[modifyProcess](#)

[runProcess](#)

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 ElectricCommander.

Argument Type: String

processType

Description: Defines the type of action performed by the process.

Argument Type: ProcessType

timeLimit

Description: Maximum amount of time that the step can execute; abort if it exceeds this time.

Argument Type: String

timeLimitUnits

Description: Units for the step- time limit: seconds, minutes, or hours.

Argument Type: TimeLimitUnits

workspaceName

Description: Name of the default workspace for this process.

Argument Type: String

Response

Returns a process component element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool createProcess default newProcess --componentName VCScomponent
```

deleteProcess

Deletes an application or component process.

Required Arguments

projectName

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

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

Optional Arguments

applicationName

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

Argument Type: String

componentApplicationName

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

Argument Type: String

componentName

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

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->deleteProcess("default", "newProcess",  
{componentName => "Component1"});
```

ectool

Syntax:

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

Example:

```
ectool deleteProcess default newProcess --componentName Component1
```

getProcess

Retrieves an application or component process.

Required Arguments

`projectName`

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

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationName`

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

Argument Type: String

`componentApplicationName`

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

Argument Type: String

`componentName`

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

Argument Type: String

Response

Retrieves the specified process element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getProcess default newProcess --componentName VCScomponent
```

getProcesses

Retrieves all processes in an application or component.

Required Arguments

projectName

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

Argument Type: String

Optional Arguments

applicationName

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

Argument Type: String

componentApplicationName

Description: Application name of the component, if the component is scoped to application.

Argument Type: String

componentName

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

Argument Type: String

Response

Retrieves zero or more process elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getProcesses default --componentName VCScomponent
```

modifyProcess

Modifies an existing process.

Required Arguments

`projectName`

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

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationName`

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

Argument Type: String

`componentApplicationName`

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

Argument Type: String

`componentName`

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

Argument Type: String

`credentialName`

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

Argument Type: String

`description`

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

Argument Type: String

`newName`

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

Argument Type: String

processType

Description: Defines the type of action performed by the process.

Argument Type: ProcessType

timeLimit

Description: Maximum amount of time that the step can execute; abort if it exceeds this time.

Argument Type: String

timeLimitUnits

Description: Units for step time limit: seconds, minutes, or hours.

Argument Type: TimeLimitUnits

workspaceName

Description: Name of the default workspace for this process.

Argument Type: String

Response

Retrieves an updated process element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyProcess("default", "newProcess", {componentName => "VCS",  
  newName => "VCScomponent", description => "An updated description"});
```

ectool

Syntax:

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

Example:

```
ectool modifyProcess default newProcess --componentName VCScomponent  
  --newName VCS --description "A description"
```

runProcess

Runs the specified process.

Required Arguments

projectName

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

Argument Type: String

applicationName

Description: Name of the application that owns the process; must be unique among all applications in the project.

Argument Type: String

processName

Description: Name of the application process.

Argument Type: String

tierMapName

Description: Name of the tier map used to determine where to run the process.

Argument Type: String

Optional Arguments

actualParameter

Description: Parameters passed as arguments to the process.

Argument Type: Map

destinationProject

Description: Project that will own the job.

Argument Type: String

priority

Description: Priority of the job.

Argument Type: JobPriority

validate

Description: Validates that the application process, tier map, and environment are well-defined and valid before the running the application process. This argument defaults to true.

Argument Type: Boolean

Response

Returns new job ID.

ec-perl

Syntax:

```
$<object>->runProcess(<projectName>, <applicationName>, <processName>,  
<tierMapName>, {<optionals>});
```

Example:

```
$ec->runProcess("default", "NewApp", "newProcess", "TierMap2",  
{destinationProject => "deploy1"});
```

ectool

Syntax:

```
ectool runProcess <projectName> <applicationName> <processName> <tierMapName>  
[optionals...]
```

Example:

```
ectool runProcess default NewApp newProcess TierMap2 --destinationProject  
deploy1
```

API Commands - Process Dependency

[createProcessDependency](#)

[deleteProcessDependency](#)

[getProcessDependencies](#)

[modifyProcessDependency](#)

createProcessDependency

Creates a dependency between two process steps.

Required Arguments

`projectName`

Description: Name of the project; must be unique among all projects.

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

`processStepName`

Description: Name of the process step.

Argument Type: String

`targetProcessStepName`

Description: Name of the target process step.

Argument Type: String

Optional Arguments

`applicationName`

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

`branchCondition`

Description: Condition of the branch.

Argument Type: String

`branchConditionName`

Description: Name of the branch condition.

Argument Type: String

branchConditionType

Description: Type of the branch condition.

Argument Type: BranchConditionType

branchType

Description: Type of the branch.

Argument Type: BranchType

componentApplicationName

Description: If specified, the component is scoped to this application not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Returns a process dependency element.

ec-perl

Syntax:

```
$<object>->createProcessDependency(<projectName>, <processName>,  
    <processStepName>, <targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->createProcessDependency("default", "newProcess", "Step C", "Step D",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool createProcessDependency <projectName> <processName> <processStepName>  
    <targetProcessStepName> [optionals...]
```

Example:

```
ectool createProcessDependency default newProcess "Step A" "Step B"  
    --componentName VCScomponent
```

deleteProcessDependency

Deletes a dependency between two process steps.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

targetProcessStepName

Description: Name of the target process step.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteProcessDependency(<projectName>, <processName>,  
    <processStepName>, <targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->deleteProcessDependency("default", "newProcess", "Step B", "Step C",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool deleteProcessDependency <projectName> <processName> <processStepName>  
    <targetProcessStepName> [optionals...]
```

Example:

```
ectool deleteProcessDependency default newProcess "Step B" "Step C"  
    --componentName VCScomponent
```

getProcessDependencies

Retrieves all dependencies for a process.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves zero or more process dependency elements.

ec-perl

Syntax:

```
$<object>->getProcessDependencies(<projectName>, <processName>,  
    {<optionals>});
```

Example:

```
$ec->getProcessDependencies("default", "newProcess",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool getProcessDependencies <projectName> <processName> [optionals...]
```

Example:

```
ectool getProcessDependencies default newProcess --componentName VCScomponent
```

modifyProcessDependency

Modifies a dependency between two process steps.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

targetProcessStepName

Description: Name of the target process step.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application.

Argument Type: String

branchCondition

Description: Condition of the branch.

Argument Type: String

branchConditionName

Description: Name of the branch condition.

Argument Type: String

branchConditionType

Description: Type of the branch condition.

Argument Type: BranchConditionType

branchType

Description: Type of the branch.

Argument Type: BranchType

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

ec-perl

Syntax:

```
$<object>->modifyProcessDependency(<projectName>, <processName>, <processStepName>,  
<targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->modifyProcessDependency("default", "newProcess", "Step1", "StepA",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool modifyProcessDependency <projectName> <processName> <processStepName>  
<targetProcessStepName> [optionals...]
```

Example:

```
ectool modifyProcessDependency default newProcess Step1 StepA --componentName  
VCScomponent
```

API Commands - Process Step

[createProcessStep](#)

[deleteProcessStep](#)

[getProcessStep](#)

[getProcessSteps](#)

[modifyProcessStep](#)

Note: Several of the following API commands contain context type optional arguments. For example, a step command may reference either a procedure or component.

createProcessStep

Creates a new process step.

Required Arguments

`projectName`

Description: Name for the project; must be unique among all projects.

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

`processStepName`

Description: Name of the process step.

Argument Type: String

Optional Arguments

`actualParameters`

Description: Actual parameters (<var1>=<val1> [<var2>=<val2> ...]) passed to an invoked subprocedure or process.

Argument Type: Map

`afterProcessStep`

Description: If specified, the process step will be placed after the named process step.

Argument Type: String

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

applicationTierName

Description: Application tier on which to run the step.

Argument Type: String

beforeProcessStep

Description: If specified, the process step will be placed before the named process step.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

credentialName

Description: Name of the credential object.

Argument Type: String

description

Description: Comment text describing this object; not interpreted at all by the ElectricCommander platform.

Argument Type: String

errorHandling

Description: Specifies error handling for this step.

Argument Type: ErrorHandling

includeCompParameterRef

Description: True if the actual parameters should be generated from component properties. Works for artifact components only.

Argument Type: Boolean

processStepType

Description: Defines type of the process step.

Argument Type: ProcessStepType

subcomponent

Description: If referencing a component process, the name of the component.

Argument Type: String

subcomponentProcess

Description: If referencing a component process, the name of the component process.

Argument Type: String

subprocedure

Description: If referencing a procedure, the name of the procedure.

Argument Type: String

subproject

Description: If referencing a procedure, the name of the procedure's project.

Argument Type: String

timeLimit

Description: Maximum amount of time that the step can execute; abort if it exceeds this time.

Argument Type: String

timeLimitUnits

Description: Units for the step time limit: seconds, minutes, or hours.

Argument Type: TimeLimitUnits

workspaceName

Description: Name of the workspace.

Argument Type: String

Response

Returns a process step element.

ec-perl

Syntax:

```
$<object>->createProcessStep(<projectName>, <processName>,  
    <processStepName>, {<optionals>});
```

Example:

```
$ec->createProcessStep("default", "newProcess", "Step 1",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool createProcessStep <projectName> <processName> <processStepName>  
    [optionals...]
```

Example:

```
ectool createProcessStep default newProcess "Step A"  
--componentName VCScomponent
```

deleteProcessStep

Deletes an application or component process step.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteProcessStep (<projectName>, <processName>,  
    <processStepName>, {<optionals>});
```

Example:

```
$ec->deleteProcessStep ("default", "newProcess", "stepToDelete",  
    {componentName=> "VCScomponent"});
```

ectool

Syntax:

```
ectool deleteProcessStep <projectName> <processName> <processStepName>  
    [optionals...]
```

Example:

```
ectool deleteProcessStep default newProcess "stepToDelete"  
    --componentName VCScomponent
```

getProcessStep

Gets an application or component process step.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

processName

Description: The name of the process.

Argument Type: String

processStepName

Description: The name of the process step.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves the specified process step element.

ec-perl

Syntax:

```
$<object>->getProcessStep(<projectName>, <processName>, <processStepName>,  
    {<optionals>});
```

Example:

```
$ec->getProcessStep("default", "newProcess", "Step 1",  
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool getProcessStep <projectName> <processName> <processStepName>  
    [optionals...]
```

Example:

```
ectool getProcessStep default newProcess "Step A"  
    --componentName VCScomponent
```

getProcessSteps

Retrieves all the process steps in an application or component process.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves zero or more process step elements.

ec-perl

Syntax:

```
$<object>->getProcessSteps(<projectName>, <processName>, {<optionals>});
```

Example:

```
$ec->getProcessSteps("default", "newProcess",  
    {componentName=> "VCscomponent"});
```

ectool

Syntax:

```
ectool getProcessSteps <projectName> <processName> [optionals...]
```

Example:

```
ectool getProcessSteps default newProcess --componentName VCscomponent
```

modifyProcessStep

Modifies an existing process step.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

Optional Arguments

`actualParameters`

Description: Actual parameters passed to an invoked subprocedure or process.

Argument Type: Map

`afterProcessStep`

Description: If specified, the process step will be placed after the named process step.

Argument Type: String

`applicationName`

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

`applicationTierName`

Description: Name of the application tier on which to run the step.

Argument Type: String

`beforeProcessStep`

Description: If specified, the process step will be placed before the named process step.

Argument Type: String

`clearActualParameters`

Description: True if the step should remove all actual parameters.

Argument Type: Boolean

`componentApplicationName`

Description: If specified, the component is scoped to this application not the project.

Argument Type: String

`componentName`

Description: Name of the component, if the process is owned by a component.

Argument Type: String

`credentialName`

Description: Name of the credential object.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted at all by the ElectricCommander platform.

Argument Type: String

`errorHandling`

Description: Specifies error handling for this step.

Argument Type: ErrorHandling

`includeCompParameterRef`

Description: True if the actual parameters should be generated from component properties. Works for artifact components only.

Argument Type: Boolean

`newName`

Description: New name for an existing object that is being renamed.

Argument Type: String

`processStepType`

Description: Defines type of the process step.

Argument Type: ProcessStepType

`subcomponent`

Description: If referencing a component process, the name of the component.

Argument Type: String

`subcomponentApplicationName`

Description: If referencing a component process, the name of the component application (if it has not been scoped to a project).

Argument Type: String

`subcomponentProcess`

Description: If referencing a component process, the name of the component process.

Argument Type: String

`subprocedure`

Description: If referencing a procedure, the name of the procedure.

Argument Type: String

`subproject`

Description: If referencing a procedure, the name of the procedure's project.

Argument Type: String`timeLimit`**Description:** Maximum amount of time that the step can execute; abort if it exceeds this time.**Argument Type:** String`timeLimitUnits`**Description:** Units for the step time limit: seconds, minutes, or hours.**Argument Type:** TimeLimitUnits`workspaceName`**Description:** Name of the workspace.**Argument Type:** String**Response**

Retrieves an updated process step element.

ec-perl

Syntax:

```
$ec->modifyProcessStep(<projectName>, <processName>,  
    <processStepName>, {<optionals>});
```

Example:

```
$ec->modifyProcessStep ("default", "newProcess", "Step 1",  
    {componentName => "VCSCcomponent", newName => "Step 2",  
    description => "A description"});
```

ectool

Syntax:

```
ectool modifyProcessStep <projectName> <processName> <processStepName>  
    [optionals...]
```

Example:

```
ectool modify ProcessStep newProcess "Step A"  
    --componentName VCSCcomponent --newName "Step B"  
    --description "A description"
```

API Commands - Project Management

[createProject](#)
[deleteProject](#)
[getProject](#)
[getProjects](#)
[modifyProject](#)

createProject

Creates a new project.

You must specify a `projectName`.

Arguments	Descriptions
<code>credentialName</code>	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<i>cred1</i>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>description</code>	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
<code>projectName</code>	This is any name of your choice for your new project.
<code>resourceName</code>	The name of the resource to use as the default for steps run by procedures in this project.
<code>workspaceName</code>	The name of a workspace to use as the default for steps run by procedures in this project.

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->createProject(<projectName>, {<optionals>});`

Example

```
$cmdr->createProject("Test Proj", {workspaceName => "Test_WS"});
```

ectool

syntax: `ectool createProject <projectName> ...`

Example

```
ectool createProject "Test Proj" --workspaceName "Test WS"
```

[Back to Top](#)

deleteProject

Deletes a project, including all procedures, procedure steps, and jobs within that project.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project you want to delete.

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteProject(<projectName>);`

Example

```
$cmdr->deleteProject("Test Proj");
```

ectool

syntax: `ectool deleteProject <projectName>`

Example

```
ectool deleteProject "Test Proj"
```

[Back to Top](#)

getProject

Finds a project by its name.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project you need to retrieve.

Positional arguments

`projectName`

Response

One `project` element.

ec-perl

syntax: `$cmdr->getProject (<projectName>);`

Example

```
$cmdr->getProject ("Test Proj");
```

ectool

syntax: `ectool getProject <projectName>`

Example

```
ectool getProject "Test Proj"
```

[Back to Top](#)

getProjects

Retrieves all projects.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [project](#) elements.

Note: This response includes all projects in the system, including plugin projects, which are not displayed on the Projects page in the web UI.

ec-perl

syntax: `$cmdr->getProjects();`

Example

```
$cmdr->getProjects();
```

ectool

syntax: `ectool getProjects`

Example

```
ectool getProjects
```

[Back to Top](#)

modifyProject

Modifies an existing project.

You must specify a `projectName`.

Arguments	Descriptions
<code>credentialName</code>	<code>credentialName</code> can be one of two forms: relative (for example, " <code>cred1</code> ") - the credential is assumed to be in the project that contains the request target object. absolute (for example, " <code>/projects/BuildProject/credentials/cred1</code> ") - the credential can be from any specified project, regardless of the target object's project.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>newName</code>	Supply any name of your choice to rename the project.

Arguments	Descriptions
projectName	The name of the project you want to modify.
resourceName	The name of the resource used as the default for steps run by procedures in this project.
workspaceName	The name of the default workspace where job output is stored.

Positional arguments

projectName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->modifyProject(<projectName>, {...});

Example

```
$cmdr->modifyProject("Test Proj", {description => "A very simple project"});
```

ectool

syntax: ectool modifyProject <projectName> ...

Example

```
ectool modifyProject "Test Proj" --description "A very simple project"
```

[Back to Top](#)

API Commands - Property Management

```
createProperty  
deleteProperty  
evalScript  
expandString  
getProperties  
getProperty  
incrementProperty  
modifyProperty  
setProperty
```

createProperty

Creates a regular string or nested property sheet using a combination of property path and context.

You must specify a `propertyName` and `locator` arguments to define where (or on which object) you are creating this property.

Note: The name "properties" is NOT a valid property name.

Arguments	Descriptions
<code>propertyName</code>	The name of the property to create. It may be a relative or absolute property path, including "my" paths such as <code>"/myProject/prop1"</code> .
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact container of the property sheet which owns the property.
<code>artifactVersionName</code>	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name—the Commander server interprets either name form correctly.

Arguments	Descriptions
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
credentialName	<p>The name of the credential container of the property sheet which owns the property.</p> <p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
description	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
expandable	<p>Whether or not the property is recursively expandable.</p> <p><Boolean flag - 0 1 true false> Determines whether the property value will undergo property expansion when it is fetched. Default is "true".</p>
extendedContextSearch	For simple property names, whether or not to search objects in the hierarchy to find the desired property.
gatewayName	The name of the gateway.
groupName	The name of the group where you want to create a property.

Arguments	Descriptions
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin where you want to create a property.
procedureName	The name of the procedure. Must be combined with its projectName.
processName	The name of the process, if the container is a process or process step
processStepName	The name of the process step, if the container is a process step.
projectName	The name of the project container of the property sheet which owns the property; must be unique among all projects.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
propertyType	<string sheet> Indicates whether to create a string property or a sub-sheet. Default is "string".
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource where you want to define the new property.
resourcePoolName	The name of a pool containing one or more resources.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.

Arguments	Descriptions
stepName	The name of the step. If you are using a step name to define the location for the new property, you must use projectName and procedureName also.
scheduleName	The name of the schedule. If you re using a schedule name to define the location for the new property, you must use projectName also.
stateDefinitionName	The name of the state definition container of the property sheet which owns the property.
stateName	The name of the state container of the property sheet which owns the property.
stepName	The name of the step container of the property sheet which owns the property.
systemObjectName	The name of the special system object. In this context, only <code>server</code> is legal.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The user name where you want to add a property.
value	For a string property (see <code>propertyType</code> above), this is the value of the property. For a sheet property, this argument is invalid.
valueFile	This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. The contents of the <i>valuefile</i> is read and stored in the "value" field for a string property. This is an alternative argument for value and is useful if the "value" field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace where you want to add a property.
zoneName	The name of the zone.

Positional arguments

`propertyName`

Response

An XML stream that echoes the new property, including its ID, which is assigned by the ElectricCommander server.

ec-perl

syntax: `$cmdr->createProperty(<propertyName>, {<optionals>});`

Examples

```
$cmdr->createProperty('/myJob/Runtime Env/PATH', {value => 'c:\bin'});
```

```
$cmdr->createProperty('Runtime Env/PATH', {value => 'c:\bin', ...});
```

ectool

syntax: `ectool createProperty <propertyName> ...`

Examples

```
ectool createProperty "/myJob/Runtime Env/PATH" --value "c:\bin"
```

```
ectool createProperty "Runtime Env/PATH" --value "c:\bin" --jobId 4fa765dd-73f1-11e3-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.

Arguments	Descriptions
<code>propertyName</code>	The name of the property to delete.
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.

Arguments	Descriptions
<code>artifactName</code>	The name of the artifact container of the property sheet which owns the property.
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "<code>groupId:artifactKey:version</code>" and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>
<code>componentName</code>	The name of the component container of the property sheet which owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	<p>Whether or not the property is recursively expandable.</p> <p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1></code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>environmentName</code>	The name of the environment container of the property sheet which owns the property; must be unique among all projects
<code>environmentTierName</code>	The name of the environment tier container of the property sheet which owns the property.

Arguments	Descriptions
extendedContextSearch	<p>For simple property names, whether or not to search objects in the hierarchy to find the desired property.</p> <p><Boolean flag -0 1 true false> If set, and there is an object specifier in the command, ElectricCommander first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:</p> <ol style="list-style-type: none"> 1. If the object specifier is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 2. If the object specifier is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. <p>Default setting is "true."</p>
gatewayName	The name of the gateway.
groupName	The name of a group that contains this property.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of a plugin that may contain a property you want to delete.
procedureName	<p>The name of the procedure containing the property you want to delete.</p> <p>Also requires <code>projectName</code></p>
processName	The name of the process, if the container is a process or process step
processStepName	The name of the process step, if the container is a process step.

Arguments	Descriptions
<code>projectName</code>	The name of the project that contains the property you want to delete.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource that contains the property you want to delete.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule containing the property you want to delete. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step containing the property you want to delete. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	The name of a special system object. Only 'sever' is legal in this context.
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The user name that contains this property.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace containing this property.
<code>zoneName</code>	The name of the zone.

Positional arguments

`propertyName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteProperty(<propertyName>, { ... });`

Example

```
$cmdr->deleteProperty("/projects/Sample project/Changeset ID");
```

ectool

syntax: `ectool deleteProperty <propertyName> ...`

Example

```
ectool deleteProperty "/projects/Sample project/Changeset ID"
```

[Back to Top](#)

evalScript

Evaluates a script in a given context. This API is similar to `expandString` except that it evaluates the `value` argument as a Javascript block, without performing any property substitution on either the script or the result. The string value of the final expression in the script is returned as the `value` element of the response.

You must specify a `value` to evaluate.

Arguments	Descriptions
<code>value</code>	The script to evaluate.
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact.
<code>artifactVersionName</code>	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.

Arguments	Descriptions
<code>componentName</code>	The name of the component container of the property sheet which owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	<p>The name of the credential container of the property sheet which owns the property.</p> <p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<i>cred1</i>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>environmentName</code>	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
<code>environmentTierName</code>	The name of the environment tier container of the property sheet which owns the property.
<code>gatewayName</code>	The name of the gateway.
<code>groupName</code>	The name of a group where you might evaluate a script.
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created.
<code>notifierName</code>	The name of the email notifier.
<code>objectId</code>	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
<code>path</code>	Property path string.
<code>pluginName</code>	The name of a plugin where you might evaluate a script.

Arguments	Descriptions
<code>procedureName</code>	The name of a procedure where you might need to evaluate a script. Also requires <code>projectName</code>
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>projectName</code>	The name of the project that contains the script to evaluate.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of a resource where you might evaluate a script.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of a schedule within this project. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step whose script you might evaluate. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	System object names include: <code>admin directory log priority projects resources server session workspaces</code>
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The name of the user where you may need to evaluate a script.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.

Arguments	Descriptions
workspaceName	The name of a workspace where you may need to evaluate a script.
zoneName	The name of the zone.

Positional arguments

value

Response

The string value of the final expression in the Javascript block inside a `value` element.

ec-perl

syntax: `$cmdr->evalScript (<value>);`

Examples

```
my $result = $ec->evalScript (q{"ip=" + server.hostIP+", name=" + server.hostName})
->findvalue("//value");
```

```
my $result = $ec->evalScript (q{myProject.projectName}, {jobId => '4fa765dd-73f1-11e3-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
```

[Back to Top](#)

expandString

Expands property references in a string, in the current context.

You must specify a `value` and a context in which to perform the expansion or a `valueFile` option.

Arguments	Descriptions
value	The string value to expand in the given context.

Arguments	Descriptions
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact.
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.</p>
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
credentialName	<p>The name of the credential container of the property sheet which owns the property.</p> <p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, <code>"cred1"</code>) - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>) - the credential can be from any specified project, regardless of the target object's project.</p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
gatewayName	The name of the gateway.
groupName	The name of a group where you might expand a string.

Arguments	Descriptions
<code>jobId</code>	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
<code>jobStepId</code>	The unique identifier for a job step, assigned automatically when the job step is created.
<code>notifierName</code>	The name of the email notifier.
<code>objectId</code>	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
<code>path</code>	Property path string.
<code>pluginName</code>	The name of a plugin where you might expand a string.
<code>procedureName</code>	The name of a procedure where you might need to expand a string. Also requires <code>projectName</code>
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>projectName</code>	The name of the project that contains the string to expand.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of a resource where you might expand a string.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of a schedule within this project. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.

Arguments	Descriptions
stepName	The name of the step whose string you might be expanding. Also requires projectName and procedureName
systemObjectName	System object names include: admin directory log priority projects resources server session workspaces
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user where you may need to expand the string.
valueFile	This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for value and is useful if the value field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of a workspace where you may need to expand the string.
zoneName	The name of the zone.

Positional arguments

value

Response

The expanded string value.

ec-perl

syntax: \$cmdr->expandString(<value>, {<optionals>});

Examples

```
$cmdr->expandString('${fullUserName}', {userName => "admin"})->findvalue('//value')  
->value();
```

```
$cmdr->expandString('${/myWorkspace/agentUncPath}/${logFileName}',  
  {jobStepId => 5da765dd-73f1-11e3-b67e-b0a420524153})->findvalue('//value')->valu  
e();
```

ectool

syntax: ectool expandString <value> ...

Examples

```
ectool expandString '${fullUserName}' --userName admin
```

```
ectool expandString '${/myWorkspace/agentUncPath}/${logFileName}'  
--jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153
```

[Back to Top](#)

getProperties

Retrieves all properties associated with an object, along with the property sheet identifier for the object's property sheet.

You must specify object locators for the properties you want to retrieve.

Arguments	Descriptions
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact container of the property sheet which owns the property.
artifactVersionName	The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name--the Commander server interprets either name form correctly.
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.

Arguments	Descriptions
credentialName	<p>The name of the credential containing the properties to retrieve. <code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Also requires <code>projectName</code></p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
expand	<Boolean flag - 0 1 true false> Default value=1 (true), which means the value of each property will be expanded. A value of "0" (false) will cause the unexpanded value of each property to be returned.
gatewayName	The name of the gateway.
groupName	The name of the group containing the properties to retrieve.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
path	The path to the property sheet containing the properties to retrieve. If the full path to the property sheet is specified, no additional object locators are needed.

Arguments	Descriptions
pluginName	The name of the plugin containing the properties to retrieve.
procedureName	The name of the procedure containing the properties to retrieve. Also requires projectName
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.
projectName	The name of the project containing the properties to retrieve.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
recurse	<i><Boolean flag - 0 1 true false></i> Default value=0 (false), which means properties of nested sheets will not be included in the response. If you want the properties from all nested sheets to be retrieved, use the value of "1" for true.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the properties to retrieve.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the properties to retrieve. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the properties to retrieve. Also requires projectName and procedureName
systemObjectName	The name of the system object containing the properties to retrieve. Only "server" is supported.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.

Arguments	Descriptions
userName	The name of the user containing the properties to retrieve.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow. Also requires <code>projectName</code>
workspaceName	The name of the workspace containing the properties to retrieve.
zoneName	The name of the zone.

Positional arguments

Arguments to locate the property, beginning with the top-level object.

Response

A [propertySheet](#) element, which contains zero or more [property](#) elements and nested `propertySheet` elements.

ec-perl

syntax: `$cmdr->getProperties({<optionals>});`

Examples

```
$cmdr->getProperties({resourceName => "r2"});
```

ectool

syntax: `ectool getProperties ...`

Examples

```
ectool getProperties --resourceName "r2"
```

[Back to Top](#)

getProperty

Retrieves the specified property value.

You must specify a `propertyName`.

Note: This specification can be the full path to the property or it can be relative to an object, which then requires appropriate object locators.

Arguments	Descriptions
propertyName	The name or path for the property to retrieve.
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact.
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</p>
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
credentialName	<p>The name of the credential containing the property to retrieve. <code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Also requires <code>projectName</code></p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.

Arguments	Descriptions
expand	<i><Boolean flag - 0 1 true false></i> Default value=1 (true), which means the value of each property will be expanded. A value of "0" (false) will cause the unexpanded value of each property to be returned.
extendedContextSearch	<p>For simple property names, whether or not to search objects in the hierarchy to find the desired property.</p> <p><i><Boolean flag - 0 1 true false></i> If set, and there is an object locator in the command, Commander first looks for the property in that object locator, but also searches in other locations if not found, according to the following rules:</p> <p>If the object locator is a procedure, Commander looks for the property in the project where the procedure resides.</p> <p>If the object locator is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.</p> <p>Default setting is "true."</p>
gatewayName	The name of the gateway.
groupName	The name of the group containing the property to retrieve.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin containing the property to retrieve.
procedureName	The name of the procedure containing the property to retrieve. Also requires <code>projectName</code>

Arguments	Descriptions
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.
projectName	The name of the project containing the property to retrieve.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the property to retrieve.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the property to retrieve. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the property to retrieve. Also requires projectName and procedureName
systemObjectName	The name of the system object containing the property to retrieve. Only "server" is supported.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user containing the property to retrieve.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property to retrieve.
zoneName	The name of the zone.

Positional arguments

propertyName

Response

A [property sheet](#) or a text string containing the value of the property.

Property value example: 35491

ec-perl

syntax: \$cmdr->getProperty(<propertyName>, {<optionals>});

Examples

```
use XML::XPath;
$cmdr->getProperty("/myProject/changeset ID")->findvalue('//value')->value();

$cmdr->getProperty("Changeset ID", {propertyName => "Sample Project"})->findvalue('/
value')->value();
```

ectool

syntax: ectool getProperty <propertyName> ...

Examples

```
ectool getProperty "/myProject/changeset ID"

ectool getProperty "Changeset ID" --projectName "Sample Project"

# Retrieve the /users/<userName>/providerName property.

ectool getProperty --objectID <ID> --propertyName "/users/<userName>/providerName"
```

[Back to Top](#)

incrementProperty

Atomically increments the specified property value by the `incrementBy` amount. If the property does not exist, it will be created with an initial value of the `incrementBy` amount.

You must specify a `propertyName` and `incrementBy`.

Arguments	Descriptions
propertyName	The name of the property to increment.
incrementBy	This is positive or negative integer.
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.

Arguments	Descriptions
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact.
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "<code>groupId:artifactKey:version</code>" and the object is searched either way you specify its name--the Commander server interprets either name form correctly.</p>
<code>componentName</code>	The name of the component container of the property sheet which owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>environmentName</code>	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
<code>environmentTierName</code>	The name of the environment tier container of the property sheet which owns the property.

Arguments	Descriptions
extendedContextSearch	<p>For simple property names, whether or not to search objects in the hierarchy to find the desired property.</p> <p><Boolean flag - 0 1 true false> If set, and there is an object specified in the command, ElectricCommander first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:</p> <ol style="list-style-type: none"> 1) If the object specifier is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 2) If the object specifier is a job step, ElectricCommander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. <p>Default setting is "true."</p>
gatewayName	The name of the gateway.
groupName	The name of the group containing the property to increment.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin containing a property to increment.
procedureName	<p>The name of the procedure containing this property.</p> <p>Also requires <code>projectName</code></p>
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.
projectName	<p>The name of the project containing this property.</p> <p>Also requires <code>procedureName</code></p>

Arguments	Descriptions
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource containing this property.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule containing this property. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step containing this property. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	Only <code>server</code> is a valid system object for this API.
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The name of the user containing this property.
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace containing this property.
<code>zoneName</code>	The name of the zone.

Positional arguments

`propertyName`, `incrementBy`

Response

A text string containing the updated numeric property value.

ec-perl

syntax: `$cmdr->incrementProperty(<propertyName> <incrementBy> ...);`

Examples

```
$cmdr->incrementProperty("Build Number", 1, {procedureName => "Delay", projectName => "Sample Project"});
```

```
$cmdr->incrementProperty("/projects/Sample Project/procedures/Delay/Build Number", 1);
```

```
$cmdr->incrementProperty("procedures/Delay/Build Number", 1, {projectName => "Sample Project"});
```

ectool

syntax: ectool incrementProperty <propertyName> <incrementBy> ...

Examples

```
ectool incrementProperty "Build Number" 1 --procedureName "Delay" --projectName "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 name "properties" is NOT a valid property name.

Arguments	Descriptions
<code>propertyName</code>	The name of the property to be modified; must be unique within the property sheet. This argument can be a path.
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact.

Arguments	Descriptions
artifactVersionName	The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name--the Commander server interprets either name form correctly.
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
credentialName	credentialName can be one of two forms: relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are: <a> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.

Arguments	Descriptions
expandable	<i><Boolean flag -0 1 true false></i> - Determines whether the property value will undergo property expansion when it is fetched. Default is "true".
extendedContextSearch	For simple property names, whether or not to search objects in the hierarchy to find the desired property. <i><Boolean flag - 0 1 true false></i> If set, and there is an object specified in the command, ElectricCommander first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules: 1) If the object specifier is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 2) If the object specifier is a job step, ElectricCommander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. Default setting is "true."
gatewayName	The name of the gateway.
groupName	The name of the group containing the property to be modified.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
newName	Supply any name of your choice to rename the property.
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
pluginName	The name of the plugin containing the property to be modified.
procedureName	The name of the procedure containing the property to be modified. Also requires <code>projectName</code>

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the property to be modified. Note that the property may be on the project itself or on a contained object, indicated by other arguments.
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>propertyType</code>	<code><string sheet></code> Indicates whether to create a string property or a sub-sheet. Default is "string".
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource containing the property to be modified.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule containing the property to be modified. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step containing the property to be modified. Also requires <code>projectName</code> and <code>procedureName</code>
<code>systemObjectName</code>	System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code>
<code>transitionDefinitionName</code>	The name of the transition definition.
<code>transitionName</code>	The name of the transition.
<code>userName</code>	The name of the user containing the property to be modified.
<code>value</code>	This can be any string you choose to add to a property.

Arguments	Descriptions
valueFile	This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for <i>value</i> and is useful if the "value" field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property to be modified.
zoneName	The name of the zone.

Positional arguments

propertyName

Response

An XML stream that echoes the modified property.

ec-perl

syntax: \$cmdr->modifyProperty(<propertyName>, {...});

Example

```
$cmdr->modifyProperty("Saved Variables", {description =>
    "Starting configuration of name/value pairs", jobId => 4fa765dd-73f1-11e3-b67
    e-b0a420524153});
```

ectool

syntax: ectool modifyProperty <propertyName> ...

Example

```
ectool modifyProperty "Saved Variables" --description "Starting configuration
of name/value pairs" --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

[Back to Top](#)

setProperty

Sets the value for the specified property.

You must specify a *propertyName* and *value*. The property name can be the full path to the property or it can be relative to an object, which then means you must use object locators to specify the property.

Arguments	Descriptions
<code>propertyName</code>	<p>The name or path of the property you want to set; must be unique within the property sheet.</p> <p>This argument can be a path.</p>
<code>value</code>	The value of the property.
<code>applicationName</code>	The name of the application container of the property sheet which owns the property; must be unique among all projects.
<code>applicationTierName</code>	The name of the application tier container of the property sheet which owns the property.
<code>artifactName</code>	The name of the artifact container of the property sheet which owns the property.
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "<code>groupId:artifactKey:version</code>" and the object is searched either way you specify its name--the Commander server interprets either name form correctly.</p>
<code>componentName</code>	The name of the component container of the property sheet which owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	<p>The name of the credential containing the property you want to set. <code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Also requires <code>projectName</code></p>

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
expandable	<i><Boolean flag - 0 1 true false></i> Default is "1" (true), which means the property value will be expanded when referenced. If you do not want the property to expand, use the value of "0"(false).
extendedContextSearch	<i><Boolean flag - 0 1 true false></i> If set, and there is an object specified in the command, ElectricCommander first looks for the property in the object specified, but also searches in other locations if not found, according to the following rules: 1) If the object specified is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 2) If the object specified is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. Default setting is "false."
gatewayName	The name of the gateway.
groupName	The name of the group containing the property you want to set.
jobId	The name of the job containing the property you want to set. The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Arguments	Descriptions
<code>jobStepId</code>	The name of the job step containing the property you want to set. The unique identifier for a job step, assigned automatically when the job step is created.
<code>objectId</code>	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
<code>notifierName</code>	The name of the email notifier.
<code>pluginName</code>	The name of the plugin containing the property you want to set.
<code>procedureName</code>	The name of the procedure containing the property you want to set. Also requires <code>projectName</code>
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>projectName</code>	The name of the project containing the property you want to set.
<code>propertySheetId</code>	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
<code>repositoryName</code>	The name of the repository for artifact management.
<code>resourceName</code>	The name of the resource containing the property you want to set.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.
<code>scheduleName</code>	The name of the schedule containing the property you want to set. Also requires <code>projectName</code>
<code>stateDefinitionName</code>	The name of the state definition.
<code>stateName</code>	The name of the state.
<code>stepName</code>	The name of the step containing the property you want to set. Also requires <code>projectName</code> and <code>procedureName</code>

Arguments	Descriptions
systemObjectName	The name of the system object containing the property you want to set. System objects include: admin artifactVersions directory emailConfigs log plugins server session workspaces
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user containing the property you want to set.
valueFile	This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for <i>value</i> and is useful if the value field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property you want to set.
zoneName	The name of the zone.

Positional arguments

propertyName, value

Response

An XML stream that echoes the property.

ec-perl

syntax: \$cmdr->setProperty(<propertyName>, <value>, {<optionals>});

Examples

```
$cmdr->setProperty("Changeset ID", "14992", {projectName => "Sample Project"});
```

```
$cmdr->setProperty("/myResource/Application Path", "c:\Program Files\Application");
```

```
$cmdr->setProperty("Application Path", "c:\Program Files\Application",  
  {resourceName => "r2"});
```

ectool

syntax: ectool setProperty <propertyName> <value> ...

Examples

```
ectool setProperty "Changeset ID" "14992" --projectName "Sample Project"
```

```
ectool setProperty "/myResource/Application Path" "c:\Program Files\Application"
```

```
ectool setProperty "Application Path" "c:\Program Files\Application"  
    --resourceName "r2"
```

[Back to Top](#)

API Commands - Resource Management

[addResourcesToPool](#)

[addResourceToEnvironmentTier](#)

[createResource](#)

[createResourcePool](#)

[deleteResource](#)

[deleteResourcePool](#)

[getResource](#)

[getResources](#)

[getResourcesInEnvironmentTier](#)

[getResourcesInPool](#)

[getResourcePool](#)

[getResourcePools](#)

[getResourceUsage](#)

[modifyResource](#)

[pingAllResources](#)

[pingResource](#)

[removeResourceFromEnvironmentTier](#)

[removeResourcesFromPool](#)

addResourcesToPool

Adds resources to a specific resource pool. A resource pool is a named group of resources.

You must specify a `resourcePoolName` and one or more resource names.

Arguments	Descriptions
<code>resourceNames</code>	The list of resources to add to the pool.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName, resourceName(s)`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->addResourcesToPool(<resourcePoolName>, {resourceName => [...]});`

Example

```
$cmdr->addResourcesToPool("pool1", { resourceName => ["resource1",
"resource2", "resource3"]});
```

ectool

syntax: `ectool addResourcesToPool <resourcePoolName> --resourceNames <resourceName1> ...`

(Note the plural form for the resourceNames option)

Example

```
ectool addResourcesToPool "Test Pool" --resourceNames Test1 Test2 Test3
```

[Back to Top](#)

addResourceToEnvironmentTier

Adds the given resource to the given environment tier.

You must specify the `resourceName`, `projectName`, `environmentName`. and `environmentTierName` arguments.

Arguments	Descriptions
<code>resourceName</code>	Name for the resource; must be unique among all resources. Argument Type: String
<code>projectName</code>	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment; must be unique among all projects. Argument Type: String

Arguments	Descriptions
<code>environmentTierName</code>	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->addResourceToEnvironmentTier(<resourceName>, <projectName>,  
    <environmentName>, <environmentTierName>);
```

Example:

```
$ec->addResourceToEnvironmentTier("Resource1", "default", "newEnv",  
    "envTier1");
```

ectool

Syntax:

```
addResourceToEnvironmentTier <resourceName> <projectName> <environmentName>  
    <environmentTierName>
```

Example:

```
ectool addResourceToEnvironmentTier Resource1 default newEnv envTier1
```

[Back to Top](#)

createResource

Creates a new resource.

Important Note: For a proxy resource, `proxyHostName` and `proxyPort` arguments refer to the proxying Commander agent.

`hostName` and `port` refer to the proxy target.

You must specify a `resourceName`.

Arguments	Descriptions
<code>artifactCacheDirectory</code>	The directory on the agent host where retrieved artifacts are stored.

Arguments	Descriptions
<code>block</code>	<Boolean flag - 0 1 true false> A newly created resource will be pinged. The "block" argument makes the <code>createResource</code> call block until the result of the ping is known. Default is "false".
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>hostName</code>	The name or IP address of the computer containing the ElectricCommander agent for this resource if it's an ordinary resource. If this is a proxy resource, this is the name or IP address of the proxy target.
<code>pools</code>	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.
<code>port</code>	The Commander agent port number for an ordinary resource. If a port number is not specified, the default agent port is used. The default agent port can be configured on the Server Settings page in the Commander Web Interface. For a proxy resource, this is the port number for the service running on the proxy target that will run commands on behalf of the Commander agent. For <code>ssh</code> , the default is 22.
<code>proxyCustomization</code>	Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.
<code>proxyHostName</code>	The name or IP address of the computer containing the Commander Agent used for a proxy resource.
<code>proxyPort</code>	The Commander agent port number for a proxy resource. See the <code>port</code> argument description for more details.
<code>proxyProtocol</code>	Protocol for communicating with the proxy target. Defaults to <code>ssh</code> . (This argument is not exposed in the Commander Web Interface at this time.)

Arguments	Descriptions
resourceDisabled	<Boolean flag - 0 1 true false> If set to 1, Commander will not start new steps on this resource. Defaults to "false".
repositoryNames	A list of one or more repository names—each repository name listed on a "new line".
resourceName	The name of the new resource you are creating.
shell	This sets a default shell for running step commands on this resource. The default is "cmd /q /c" for a Windows agent and "sh -e" for a UNIX agent.
stepLimit	Limits the number of steps that can run on the resource at one time. Setting the limit to 1 enforces serial access to the resource.
trusted	<Boolean flag - 0 1 true false> If "true", the resource is <i>trusted</i> . A trusted agent is one that has been "certificate verified." Agents can be either <i>trusted</i> or <i>untrusted</i> : <ul style="list-style-type: none">• trusted - the Commander server verifies the agent's identity using SSL certificate verification.• untrusted - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.
useSSL	<Boolean flag - 0 1 true false> Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true".
workspaceName	The name of the workspace this resource will use.
zoneName	The name of the zone where this resource resides.

Positional arguments

resourceName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->createResource(<resourceName>, {<optionals>});

Example

```
$cmdr->createResource("Test Resource 1", {hostname => "localhost", pools => "P1 P2"});
```

ectool

syntax: ectool createResource <resourceName> ...

Example

```
ectool createResource "Test Resource 1" --hostname localhost --pools "P1 P2"
```

[Back to Top](#)

createResourcePool

Creates a new pool for resources.

You must specify a `resourcePoolName`.

Arguments	Descriptions
<code>autoDelete</code>	<Boolean flag - 0 1 true false> - If true, the resource pool will be deleted automatically when the last resource is removed from the pool.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>resourceNames</code>	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.
<code>orderingFilter</code>	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.
<code>resourcePoolDisabled</code>	<Boolean flag - 0 1 true false> - If true, any runnable steps that refer to the pool will block until the pool is re-enabled.

Arguments	Descriptions
<code>resourcePoolName</code>	Choose any unique name for your resource pool.

Positional arguments

`resourcePoolName`

Response

Returns a `resourcePool` object.

ec-perl

syntax: `$cmdr->createResourcePool (<resourcePoolName>, {<optionals>});`

Example

```
$cmdr->createResourcePool ("aPool", {resourceName => ["resource1", "resource2"]});
```

ectool

syntax: `ectool createResourcePool <resourcePoolName> ...`

Example

```
ectool createResourcePool aPool --resourceNames resource1 resource2
```

[Back to Top](#)

deleteResource

Deletes a resource.

You must supply a `resourceName`.

Arguments	Descriptions
<code>resourceName</code>	The name of the resource to delete.

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteResource (<resourceName>);`

Example

```
$cmdr->deleteResource("Test Resource 1");
```

ectool

syntax: ectool deleteResource <resourceName>

Example

```
ectool deleteResource "Test Resource 1"
```

[Back to Top](#)

deleteResourcePool

Deletes a resource pool.

You must supply a `resourcePoolName`.

Arguments	Descriptions
<code>resourcePoolName</code>	The name of a pool (containing one or more resources) to delete.

Positional arguments

`resourcePoolName`

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteResourcePool(<resourcePoolName>);

Example

```
$cmdr->deleteResourcePool("Test Resource 1");
```

ectool

syntax: ectool deleteResourcePool <resourcePoolName>

Example

```
ectool deleteResourcePool "Test Resource 1"
```

[Back to Top](#)

getResource

Retrieves a resource by its name.

You must specify `resourceName`.

Arguments	Descriptions
<code>resourceName</code>	The name of the resource to retrieve.

Positional arguments

`resourceName`

Response

One [resource](#) element, which includes the resource ID, name, agent state, time created, host name, owner, port, disabled flag, shell, step limit, workspace name, and more. If using zones and gateways, `getResource` returns a list of gateways where this resource participates.

ec-perl

syntax: `$cmdr->getResource (<resourceName>);`

Example

```
$cmdr->getResource("Test Resource 1");
```

ectool

syntax: `ectool getResource <resourceName>`

Example

```
ectool getResource "Test Resource 1"
```

[Back to Top](#)

getResources

Retrieves all resources.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [resource](#) elements.

ec-perl

syntax: `$cmdr->getResources();`

Example

```
$cmdr->getResources();
```

ectool

syntax: ectool getResources

Example

```
ectool getResources
```

[Back to Top](#)

getResourcesInEnvironmentTier

Returns the list of resources in an environment tier.

You must specify the `projectName`, `environmentName`, and `environmentTierName` arguments.

Arguments	Descriptions
<code>projectName</code>	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment; must be unique among all projects. Argument Type: String
<code>environmentTierName</code>	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

Retrieves zero or more resource elements in the specified environment tier.

ec-perl

Syntax:

```
$<object>->getResourcesInEnvironmentTier(<projectName>, <environmentName>,  
    <environmentTierName>);
```

Example:

```
$ec->getResourcesInEnvironmentTier("default", "newEnv", "envTier1");
```

ectool

Syntax:

```
getResourcesInEnvironmentTier <projectName> <environmentName>
                                <environmentTierName>
```

Example:

```
ectool getResourcesInEnvironmentTier default newEnv envTier1
```

[Back to Top](#)

getResourcesInPool

Retrieves a list of resources in a pool.

You must specify a `pool` (name).

Arguments	Descriptions
<code>jobStepId</code>	The ID number of the job step related to this pool.
<code>pool</code>	The name of a pool containing one or more resources.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.

Positional arguments

`pool`

Response

An XML stream containing zero or more [resource](#) elements.

ec-perl

syntax: `$cmdr->getResourcesInPool (<pool>);`

Example

```
$cmdr->getResourcesInPool ("WindowsPool");
```

ectool

syntax: `ectool getResourcesInPool <pool>`

Example

```
ectool getResourcesInPool WindowsPool
```

[Back to Top](#)

getResourcePool

Retrieves a specified resource pool by name.

You must specify a `resourcePoolName`.

Arguments	Descriptions
<code>resourcePoolName</code>	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

An XML stream containing one `resourcePool` element.

ec-perl

syntax: `$cmdr->getResourcePool (<resourcePoolName>);`

Example

```
$cmdr->getResourcePool ("WindowsPool");
```

ectool

syntax: `ectool getResourcePool <resourcePoolName>`

Example

```
ectool getResourcePool WindowsPool
```

[Back to Top](#)

getResourcePools

Retrieves a list of resource pools.

Arguments	Descriptions
None	

Positional arguments

None

Response

An XML stream containing zero or more `resourcePool` elements.

ec-perl

syntax: \$cmdr->getResourcePools;

Example

```
$cmdr->getResourcePools;
```

ectool

syntax: ectool getResourcePools

Example

```
ectool getResourcePools
```

[Back to Top](#)

getResourceUsage

Retrieves resource usage information.

Arguments	Descriptions
None	

Positional arguments

None

Response

An XML stream containing zero or more [resourceUsage](#) elements.

ec-perl

syntax: \$cmdr->getResourceUsage;

Example

```
$cmdr->getResourceUsage;
```

ectool

syntax: ectool getResourceUsage

Example

```
ectool getResourceUsage
```

[Back to Top](#)

modifyResource

Modifies an existing resource.

You must specify a `resourceName`.

Important note: For a proxy resource, `proxyHostName` and `proxyPort` arguments refer to the proxying Commander agent. `hostName` and `port` refer to the proxy target.

Arguments	Descriptions
<code>artifactCacheDirectory</code>	The directory on the agent host where retrieved artifacts are stored.
<code>block</code>	<Boolean flag - 0 1 true false> A newly modified resource will be pinged. The "block" argument makes the <code>modifyResource</code> call "block" until the result of the ping is known. Default is "false".
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>hostName</code>	The name or IP address for the ElectricCommander machine containing the agent for this resource.
<code>newName</code>	Supply any name of your choice to rename the resource.
<code>pools</code>	A space-separated list of one or more pool names where this resource is a member. The pool name can be used in place of a single resource name. ElectricCommander chooses a resource from the pool when it executes the job step.
<code>port</code>	The port number for the ElectricCommander agent. Default is to the default agent port, but you can change this port number because of port conflicts or multiple agents running on the same machine.
<code>proxyCustomization</code>	Perl code customizing how the proxy resource communicates with the proxy target. Only applicable for proxy resources.
<code>proxyHostName</code>	The IP address of the computer containing the ElectricCommander Agent used for a proxy resource.
<code>proxyPort</code>	The Commander agent port number for a proxy resource. See the <code>port</code> argument for more details.

Arguments	Descriptions
<code>proxyProtocol</code>	Protocol for communicating with the proxy target. Defaults to <code>ssh</code> . This argument is not exposed in the Commander web interface at this time.
<code>repositoryNames</code>	A list of repository names with each repository name listed on a "new line".
<code>resourceDisabled</code>	<Boolean flag - 0 1 true false> If set to 1, ElectricCommander will not start new steps on this resource.
<code>resourceName</code>	The name of the resource being modified.
<code>shell</code>	This sets a default shell for running step commands on this resource. The default is " <code>cmd /q /c</code> " for a Windows agent and " <code>sh -e</code> " for a UNIX agent.
<code>stepLimit</code>	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.
<code>trusted</code>	<p><Boolean flag - 0 1 true false> If "true", the resource is <i>trusted</i>. A trusted agent is one that has been "certificate verified."</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • <i>trusted</i> - the Commander server verifies the agent's identity using SSL certificate verification. • <i>untrusted</i> - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.
<code>useSSL</code>	<p><Boolean flag - 0 1 true false> Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers.</p> <p>Default is "true".</p>
<code>workspaceName</code>	The name of the default workspace where job output is stored.
<code>zoneName</code>	The name of the zone where this resource resides.

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->modifyResource(<resourceName>, {...});

Example

```
$cmdr->modifyResource("Test Resource 1", {stepLimit => 5, shell => "bash"});
```

ectool

syntax: ectool modifyResource <resourceName> ...

Example

```
ectool modifyResource "Test Resource 1" --stepLimit 5 --shell "bash"
```

[Back to Top](#)

modifyResourcePool

Modifies an existing resource pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
autoDelete	<Boolean flag - 0 1 true false> - If true, the resource pool will be deleted automatically when the last resource is removed from the pool.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
newName	Any new unique name you choose to rename this resource pool.
resourceNames	A list of resource names to add to the pool. This value does not need to refer to an existing resource. Any names that do not resolve to an existing resource will be skipped when assigning resources to steps.

Arguments	Descriptions
<code>orderingFilter</code>	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.
<code>resourcePoolDisabled</code>	<i><Boolean flag - 0 1 true false></i> - If true, any runnable steps that refer to the pool will block until the pool is re-enabled.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

The modified `resourcePool` object.

ec-perl

syntax: `$cmdr->modifyResourcePool(<resourcePoolName>, {<optionals>});`

Example

```
$cmdr->modifyResourcePool("WindowsPool", { resourcePoolDisabled => 1});
```

ectool

syntax: `ectool modifyResourcePool <resourcePoolName> ...`

Example

```
ectool modifyResourcePool WindowsPool --resourcePoolDisabled 1
```

[Back to Top](#)

pingAllResources

Pings all resources.

Arguments	Description
<code>block</code>	<i><Boolean flag - 0 1 true false></i> Default value="0" (false), which means the call will return immediately. If you want the call to wait for responses from every resource before returning, use the value of "1"(true).

Positional arguments

None

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->pingAllResources({<optionals>});`

Example

```
$cmdr->pingAllResources();
```

ectool

syntax: `ectool pingAllResources...`

Example

```
ectool pingAllResources
```

[Back to Top](#)

pingResource

Pings one resources.

You must specify a `resourceName`.

Arguments	Descriptions
<code>resourceName</code>	The name of the resource to ping.

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->pingResource(<resourceName>);`

Example

```
$cmdr->pingResource("Test Resource 1");
```

ectool

syntax: `ectool pingResource <resourceName> ...`

Example

```
ectool pingResource "Test Resource 1"
```

[Back to Top](#)

removeResourceFromEnvironmentTier

Removes the given resource from the given environment tier.

You must specify the `resourceName`, `projectName`, `environmentName`. and `environmentTierName` arguments.

Arguments	Descriptions
<code>resourceName</code>	Name for the resource; must be unique among all resources. Argument Type: String
<code>projectName</code>	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment; must be unique among all projects. Argument Type: String
<code>environmentTierName</code>	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->removeResourceFromEnvironmentTier(<resourceName>, <projectName>,  
    <environmentName>, <environmentTierName>);
```

Example:

```
$ec->removeResourceFromEnvironmentTier("Resource"1, "default", "newEnv",  
    "envTier1");
```

ectool

Syntax:

```
removeResourceFromEnvironmentTier <resourceName> <projectName>  
    <environmentName> <environmentTierName>
```

Example:

```
ectool removeResourceFromEnvironmentTier Resource1 default newEnv envTier1
```

[Back to Top](#)

removeResourcesFromPool

Removes resources from a specified resource pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
<code>resourceNames</code>	The list of resources to remove from this pool.
<code>resourcePoolName</code>	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->removeResourcesFromPool(<resourcePoolName>, {<optionals>});`

Example

```
$cmdr->removeResourcesFromPool("Test Pool", {resourceNames => ["Test1", "Test2", "Test3"]});
```

ectool

syntax: `ectool removeResourcesFromPool <resourcePoolName> ...`

Example

```
ectool removeResourcesFromPool "Test Pool" --resourceNames Test1 Test2 Test3
```

[Back to Top](#)

API Commands - Schedule Management

```
createSchedule
deleteSchedule
getSchedule
getSchedules
modifySchedule
```

createSchedule

Creates a new schedule.

Note: If both `startTime` and `stopTime` are specified, `intervalUnits` and `interval` are used to specify an interval time to repeat running the procedure.

You must specify a `projectName` and `scheduleName`.

Arguments	Descriptions
<code>actualParameter</code>	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.
<code>beginDate</code>	<yyyy-mm-dd> The date you want the schedule to begin.
<code>credentialName</code>	The name of the credential to use for user impersonation when running the procedure. <code>credentialName</code> can be one of two forms: relative (for example, " <code>cred1</code> ") - the credential is assumed to be in the project that contains the request target object. absolute (for example, " <code>/projects/BuildProject/credentials/cred1</code> ") - the credential can be from any specified project, regardless of the target object's project.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are: <a> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>

Arguments	Descriptions
endDate	<yyyy-mm-dd> The date you want this schedule to end.
interval	Determines the repeat interval for starting new jobs.
intervalUnits	Specifies the units for the <code>interval</code> argument <hours minutes seconds continuous> If set to continuous, Commander creates a new job as soon as the previous job completes.
misfirePolicy	<ignore runOnce> Specifies the misfire policy. A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down. When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately. Defaults to "ignore".
monthDays	Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.
priority	<low normal high highest> Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.
procedureName	The procedure to run when the schedule is invoked.
projectName	The name of the project that contains the procedure this schedule will run.
scheduleDisabled	<Boolean flag - 0 1 true false> If set to 1, Commander will not start any new jobs from the schedule. Defaults to "false".
scheduleName	This is any name of your choice for this schedule.
startTime	Enter hours and minutes, formatted <code>hh:mm</code> , using the 24-hour clock. Using this schedule, ElectricCommander starts creating jobs at this time on the specified days.

Arguments	Descriptions
<code>stopTime</code>	Enter hours and minutes, formatted <code>hh:mm</code> , using the 24-hour clock. Commander stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, ElectricCommander creates one job only on each specified day.
<code>timeZone</code>	Supply the time zone (string) you want to use for this schedule.
<code>weekDays</code>	Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names "Monday", "Tuesday", and so on.

Positional arguments

`projectName`, `scheduleName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->createSchedule(<projectName>, <scheduleName>, {<optionals>});`

Example

```
$cmdr->createSchedule('Sample Project', 'Weekend', {startTime => '00:00',
    stopTime => '23:59',
    weekDays => 'Saturday Sunday',
    interval => 1,
    intervalUnits => 'hours',
    actualParameter => [{actualParameterName => 'param1', value => 'value1'}] });
```

ectool

syntax: `ectool createSchedule <projectName> <scheduleName> ...`

Example

```
ectool createSchedule "Sample Project" "Weekend" --startTime 00:00
--stopTime 23:59 --weekDays "Saturday Sunday" --interval 1 --intervalUnits hours
```

[Back to Top](#)

deleteSchedule

Deletes a schedule.

You must specify a `projectName` and `scheduleName`.

Arguments	Descriptions
projectName	The schedule you want to delete belongs to this project.
scheduleName	The name of the schedule you want to delete.

Positional arguments

projectName, scheduleName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteSchedule(<projectName>, <scheduleName>);

Example

```
$cmdr->deleteSchedule("Sample Project", "Weekend");
```

ectool

syntax: ectool deleteSchedule <projectName> <scheduleName>

Example

```
ectool deleteSchedule "Sample Project" "Weekend"
```

[Back to Top](#)

getSchedule

Retrieves a schedule by its name.

You must specify a projectName and scheduleName.

Arguments	Descriptions
projectName	The name of the project that contains the schedule to retrieve.
scheduleName	The name of the schedule to retrieve.

Positional arguments

projectName, scheduleName

Response

One [schedule](#) element.

ec-perl

syntax: `$cmdr->getSchedule(<projectName>, <scheduleName>);`

Example

```
$cmdr->getSchedule("Sample Project", "Build Schedule");
```

ectool

syntax: `ectool getSchedule <projectName> <scheduleName>`

Example

```
ectool getSchedule "Sample Project" "Build Schedule"
```

[Back to Top](#)

getSchedules

Retrieves all schedules.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the schedules to retrieve.

Positional arguments

`projectName`

Response

Zero or more [schedule](#) elements for all schedules within the named project.

ec-perl

syntax: `$cmdr->getSchedules(<projectName >);`

Example

```
$cmdr->getSchedules("Sample Project");
```

ectool

syntax: `ectool getSchedules <projectName>`

Example

```
ectool getSchedules "Sample Project"
```

[Back to Top](#)

modifySchedule

Modifies an existing schedule.

You must specify a `projectName` and a `scheduleName`.

Note: If both `startTime` and `stopTime` are specified, `intervalUnits` and `interval` are used to specify an interval to repeat running the procedure.

Arguments	Descriptions
<code>actualParameter</code>	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.
<code>beginDate</code>	<code><yyyy-mm-dd></code> The date you want the schedule to begin.
<code>clearActualParameters</code>	<code><Boolean flag - 0 1 true false></code> If set to true, all actual parameters will be removed from the schedule.
<code>credentialName</code>	The name of the credential to use for user impersonation when running the procedure. <code>credentialName</code> can be one of two forms: relative (for example, " <code>cred1</code> ") - the credential is assumed to be in the project that contains the request target object. absolute (for example, " <code>/projects/BuildProject/credentials/cred1</code> ") - the credential can be from any specified project, regardless of the target object's project.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>endDate</code>	<code><yyyy-mm-dd></code> The date you want this schedule to end.
<code>interval</code>	Determines the repeat interval for starting new jobs.

Arguments	Descriptions
<code>intervalUnits</code>	Specifies the units for the <code>interval</code> argument <hours minutes seconds continuous>. If set to <code>continuous</code> , Commander creates a new job as soon as the previous job completes.
<code>misfirePolicy</code>	<ignore runOnce> Specifies the misfire policy. A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down. When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately. Defaults to "ignore".
<code>monthDays</code>	Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.
<code>newName</code>	Supply any name of your choice to rename the schedule.
<code>priority</code>	<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.
<code>procedureName</code>	The name of the procedure to run when the schedule is invoked.
<code>projectName</code>	The name of the project containing the schedule to modify.
<code>scheduleDisabled</code>	<Boolean flag - 0 1 true false> If set to 1, Commander will not start any new jobs from the schedule.
<code>scheduleName</code>	The name of the schedule to modify.
<code>startTime</code>	Enter hours and minutes, formatted <code>hh:mm</code> , using the 24-hour clock. Commander starts creating jobs at this time on the days specified.

Arguments	Descriptions
stopTime	Enter hours and minutes, formatted <code>hh:mm</code> , using the 24-hour clock. Commander stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, Commander creates one job only on each specified day.
timeZone	Supply the time zone you want to use for this schedule.
weekDays	Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names "Monday", "Tuesday", and so on.

Positional arguments

projectName, scheduleName

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->modifySchedule(<projectName>, <scheduleName>, {...});`

Example

```
$cmdr->modifySchedule("Sample Project", "Weekend",
    {procedureName => "Delay",
     actualParameter => {actualParameterName => "Delay Time",
                        value => "5"}});
```

ectool

syntax: `ectool modifySchedule <projectName> <scheduleName> ...`

Example

```
ectool modifySchedule "Sample Project" "Weekend" --procedureName "Delay"
--actualParameter "Delay Time=5"
```

[Back to Top](#)

API Commands - Server Management

`getVersions`
`shutdownServer`
`importLicenseData`
`getAdminLicense`
`getLicense`
`getLicenses`
`getLicenseUsage`
`deleteLicense`
`getServerStatus`

getVersions

Retrieves server version information.

Arguments	Descriptions
None	

Positional arguments

None

Response

A `serverVersion` element.

ec-perl

syntax: `$cmdr->getVersions();`

Example

```
$cmdr->getVersions();
```

ectool

syntax: `ectool getVersions`

Example

```
ectool getVersions
```

[Back to Top](#)

shutdownServer

Shuts down the ElectricCommander server. Shutting down the server can take as long as a couple of minutes, depending on the server activity level at the time the shutdown command is issued.

The Commander server is composed of two processes. The main process is a Java Virtual Machine (JVM). The second process, called the "wrapper", is responsible for interacting with the native operating system as a service. This wrapper process is responsible for starting and stopping the main JVM process.

Arguments	Descriptions
<code>force</code>	<Boolean flag - 0 1 true false> The "1" flag tells the Commander server to exit immediately, without performing any of the usual associated cleanup activities. This action "kills" all running jobs.
<code>restart</code>	<Boolean flag - 0 1 true false> The "1" flag tells the Commander server to shut down normally and immediately start again.

Positional arguments

None

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->shutdownServer({<optionals>});`

Example

```
$cmdr->shutdownServer({restart => 1});
```

ectool

syntax: `ectool shutdownServer ...`

Example

```
ectool shutdownServer --restart 1
```

[Back to Top](#)

importLicenseData

Imports one or more licenses.

You must specify `licenseData`.

Arguments	Descriptions
<code>licenseData</code>	The content of a license file (perl XML API).
<code>licenseFile</code>	<code><localFileName></code> The license file to import. This is a local file that will be read by ectool. The contents is sent as the <code>licenseData</code> argument (ectool only).

Positional arguments

`licenseData`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->importLicenseData (<licenseData>)`

Example

```
my $data = 'cat license.xml';
$cmdr->importLicenseData ($data);
```

ectool

syntax: `ectool importLicenseData <licenseData>`

Example

```
ectool importLicenseData license.xml
```

[Back to Top](#)

getAdminLicense

Retrieves the admin license, which can be used when all concurrent user licenses are in use.

Arguments	Descriptions
None	

Positional arguments

None

Response

You can receive one or more responses, depending on how you are licensed and actual license usage at the time of your query.

Response examples:

When the user does not have the necessary permission to use the Administrator license:

```
<error requestId="1">
  <code>AccessDenied</code>
  <where></where>
  <message>Principal 'bob@company.com' does not have execute privileges on
    systemObject[name=licensing,id=10]</message>
  <details></details>
</error>
```

When the user has permission to get/use the Administrator license, but already has a User license:

```
<result>User 'bob@company.com@192.168.17.217' already has an active
license.</result>
```

When the user has permission to use/get the Administrator license, has no other license, and the Administrator license is not currently assigned:

```
<result>User 'bob@company.com@192.168.17.217' was given the admin
license.</result>
```

When the user has permission to get/use the Administrator license, has no license, and the Administrator license is currently assigned to someone else:

```
<result>User 'joedoe@company.com@192.168.17.217' was given the admin license
that
  previously belonged to 'bob@company.com@192.168.17.217'. </result>
```

ec-perl

syntax: \$cmdr->getAdminLicense();

Example

```
$cmdr->getAdminLicense();
```

ectool

syntax: ectool getAdminLicense

Example

```
ectool getAdminLicense
```

[Back to Top](#)

getLicense

Retrieves information for one license.

You must specify the **productName** and **featureName**.

Arguments	Descriptions
<code>featureName</code>	The name of the licensed feature. Possible features include: <code>Server</code>
<code>productName</code>	The name of the product with the licensed feature. Possible products include: <code>ElectricCommander</code>

Positional arguments

`productName`, `featureName`

Response

One [license](#) element.

ec-perl

syntax: `$cmdr->getLicense(<productName>, <featureName>);`

Example

```
$cmdr->getLicense('ElectricCommander', 'Server');
```

ectool

syntax: `ectool getLicense <productName> <featureName>`

Example

```
ectool getLicense ElectricCommander Server
```

[Back to Top](#)

getLicenses

Retrieves all license data.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [license](#) elements.

ec-perl

syntax: \$cmdr->getLicenses();

Example

```
$cmdr->getLicenses();
```

ectool

syntax: ectool getLicenses

Example

```
ectool getLicenses
```

[Back to Top](#)

getLicenseUsage

Retrieves the current license usage.

Arguments	Descriptions
None	

Positional arguments

None

Response

You may receive one or more responses for [licenseUsage](#), depending on how you are licensed and actual license usage at the time of your query.

ec-perl

syntax: \$cmdr->getLicenseUsage();

Example

```
$cmdr->getLicenseUsage();
```

ectool

syntax: ectool getLicenseUsage

Example

```
ectool getLicenseUsage
```

[Back to Top](#)

deleteLicense

Deletes a license.

You must specify a `productName` and `featureName`.

Arguments	Descriptions
<code>featureName</code>	The name of the licensed feature. Possible features include: <code>Server</code>
<code>productName</code>	The name of the product with the licensed feature. Possible products include: <code>ElectricCommander</code>

Positional arguments

`productName`, `featureName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteLicense(<productName>, <featureName>);`

Example

```
$cmdr->deleteLicense("ElectricCommander", "Server");
```

ectool

syntax: `ectool deleteLicense <productName> <featureName>`

Example

```
ectool deleteLicense ElectricCommander Server
```

[Back to Top](#)

getServerStatus

Retrieves the current status of the ElectricCommander server.

Arguments	Descriptions
<code>block</code>	<Boolean flag - 0 1 true false> A "1" flag causes the API to wait until the server reaches a "terminal state". Terminal states include <code>running</code> , <code>failed</code> , <code>stopping</code> , and <code>importFailed</code> .

Arguments	Descriptions
diagnostics	<p><Boolean flag - 0 1 true false> This argument supplies the following diagnostics" information in your output:</p> <ul style="list-style-type: none"> • <code>threadDump</code> - stack dumps of all threads in the server • <code>statistics</code> - output from all system timers • <code>systemProperties</code> - values of all java system properties • <code>environmentVariables</code> - values of all environment variables • <code>settings</code> - values of all server settings • <code>serverInfo</code> - output from <code>getServerInfo</code> call.
serverStateOnly	<p><Boolean flag - 0 1 true false> A "1" flag causes the API to limit the response to the short form, and causes ectool to return only the value of the <code>serverStatus</code> element as a simple string value.</p>
timeout	<p>This flag specifies the timeout for the <code>element</code> flag. The default value is 120 seconds.</p>

Positional arguments

None

Response

Returns the current status of the server, including the log message generated during the startup sequence.

This command returns different information depending on when and how it is called.

Note: You will get a lengthy response if you connect with a session that has admin privileges or if the server is still in a bootstrap state. After the server enters the "running" state, it is able to perform access checks but displays only the short form until you log in.

A simple response:

```
<serverState>running</serverState>
```

For more detailed server status response information, click [here](#).

ec-perl

syntax: `$cmdr->getServerStatus({<optionals>});`

Examples

```
$cmdr->getServerStatus();
```

```
$cmdr->getServerStatus({diagnostics=>1});
```

ectool

syntax:ectool getServerStatus

Examples

```
ectool getServerStatus
```

```
ectool getServerStatus --diagnostics 1
```

[Back to Top](#)

API Commands - Tier Map

[createTierMap](#)

[deleteTierMap](#)

[deleteTierMapping](#)

[getTierMaps](#)

[modifyTierMap](#)

createTierMap

Creates a new tier map for an application.

Required Arguments

`projectName`

Description: Name for the project; must be unique among all projects.

Argument Type: String

`applicationName`

Description: Name of the application; must be unique among all applications in the project.

Argument Type: String

`environmentProjectName`

Description: Name of the environment's project; must be unique among all projects.

Argument Type: String

`environmentName`

Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

Optional Arguments

`tierMapName`

Description: The name of the tier map. If not specified, the operation will generate a name of the form as follows: <applicationName>-<environmentName>.

Argument Type: String

`tierMapping`

Description: List of mappings between the application tiers and the environment tiers. The list shows the mappings as <applicationTier>=<environmentTier>.

Argument Type: Map

Response

Returns a tier-map element.

ec-perl

Syntax:

```
$<object>->createTierMap(<projectName>, <applicationName>,  
    <environmentProjectName>, <environmentName>), {<optionals>});
```

Example:

```
$ec->createTierMap("default", "newApp", "defaultEnv", "Env1",  
    {tierMapping => [{applicationTier => "AppTier1",  
        environmentTier => "EnvTier1"}], {applicationTier => "AppTier2",  
        environmentTier => "EnvTier2"}}, tierMapName => "TierMap1");
```

ectool

Syntax:

```
ectool createTierMap <projectName> <applicationName>  
    <environmentProjectName> <environmentName> [optionals...]
```

Example:

```
ectool createTierMap default newApp defaultEnv Env1 --tierMapName TierMap1  
--tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
```

deleteTierMap

Deletes a tier map from an application.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

applicationName

Description: Name of the application; must be unique among all applications in the project.

Argument Type: String

environmentProjectName

Description: Name of the environment's project; must be unique among all projects.

Argument Type: String

environmentName

Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteTierMap(<projectName>, <applicationName>,  
    <environmentProjectName>, <environmentName>);
```

Example:

```
$ec->deleteTierMap("default", "App1", "MyProj", "Env1");
```

ectool

Syntax:

```
ectool deleteTierMap <projectName> <applicationName>  
    <environmentProjectName> <environmentName>
```

Example:

```
ectool deleteTierMap default TierMapToDelete defaultEnv Env1
```

deleteTierMapping

Deletes a tier mapping from a tier map.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

applicationName

Description: Name of the application; must be unique among all applications in the project.

Argument Type: String

environmentProjectName

Description: Name of the environment's project; must be unique among all projects.

Argument Type: String

environmentName

Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

applicationTierName

Description: Name of the application tier.

Argument Type: String

Optional Arguments

None

Response

Deletes the specified tier mapping.

ec-perl

Syntax:

```
$<object>->deleteTierMapping(<projectName>, <applicationName>,  
    <environmentProjectName>, <environmentName>, <applicationTierName>);
```

Example:

```
$ec->deleteTierMap("default", "App1", "MyProj", "Env1",  
    "InstallTier");
```

ectool

Syntax:

```
ectool deleteTierMapping <projectName> <applicationName>  
    <environmentProjectName> <environmentName> <applicationTierName>
```

Example:

```
ectool deleteTierMapping default TierMapToDelete defaultEnv Env1 InstallTier
```

getTierMaps

Retrieves all tier maps that are used by the given application.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

applicationName

Description: Name of the application; must be unique among all projects.

Argument Type: String

Optional Arguments

None

Response

Returns a list of tier maps.

ec-perl

Syntax:

```
$<object>->getTierMaps(<projectName>, <applicationName>);
```

Example:

```
$ec->getTierMaps("default", "NewApp");
```

ectool

Syntax:

```
ectool getTierMaps <projectName> <applicationName>
```

Example:

```
ectool getTierMaps default NewApp
```

modifyTierMap

Modifies an existing tier map.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

applicationName

Description: Name of the application; must be unique among all applications in the project.

Argument Type: String

environmentProjectName

Description: Name of the environment's project; must be unique among all projects.

Argument Type: String

environmentName

Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

Optional Arguments

tierMapName

Description: New name of the tier map, if specified.

Argument Type: String

tierMapping

Description: List of mappings between the application tiers and the environment tiers. The list shows the mappings as <applicationTier>=<environmentTier>.

If you use this argument, new tier mappings are added or existing mappings are updated for the specified application tiers. This argument does *not* replace all the mappings and thus does *not* remove the mappings that were not specified in the API call. To remove mappings, use the ***deleteTierMapping*** command.

Argument Type: Map

Response

Retrieves the updated tier map.

ec-perl

Syntax:

```
$<object>->modifyTierMap(<projectName>, <applicationName>,  
    <environmentProjectName>, <environmentName>), {<optionals>});
```

Example:

```
$ec->modifyTierMap("default", "newApp", "defaultEnv", "Env1",  
    {tierMapping => [{applicationTier => "AppTier1",  
        environmentTier => "EnvTier1"}], {applicationTier => "AppTier2",  
        environmentTier => "EnvTier2"}}, tierMapName => "TierMap1");
```

ectool

Syntax:

```
ectool modifyTierMap <projectName> <applicationName>  
    <environmentProjectName> <environmentName> [optionals...]
```

Example:

```
ectool modifyTierMap default newApp defaultEnv Env1 --tierMapName TierMap1  
--tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
```

API Commands - User/Group Management

login	addUsersToGroup	createUser
logout	deleteUser	
createGroup	getUser	
deleteGroup	getUsers	
getGroup	modifyUser	
getGroups	removeUsersFromGroup	
modifyGroup		

addUsersToGroup

Adds ones or more specified users to a particular group.

You must specify a `groupName` and one or more user names.

Arguments	Descriptions
groupName	The name of the group you are modifying.
userNames	The list of user names to add to this group.

Positional arguments

groupName, userNames

Response

None or status OK message.

ec-perl

syntax: \$cmdr->addUsersToGroup(<groupName>, {userName=>[<userName1>, ...]});

Example

```
$cmdr->addUsersToGroup("Developers", {userName => ["John", "Jim", "Joey"]});
```

ectool

syntax: ectool addUsersToGroup <groupName> --userNames <userName1> ...

(Note the plural form for the `userNames` option)

Example

```
ectool addUsersToGroup Developers --userNames John Jim Joey
```

[Back to Top](#)

createGroup

Creates a new local group of users.

You must specify a `groupName`.

Arguments	Descriptions
<code>groupName</code>	A name you choose for the new group you are creating.
<code>userNames</code>	One or more user names to add to the group.

Positional arguments

`groupName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->createGroup(<groupName>, {<optionals>});`

Example

```
$cmdr->createGroup("Build Users", {userName => ["aallen", "Betty Barker", "cclark"]});
```

ectool

syntax: `ectool createGroup <groupName> --userNames <user1> ...`

(Note the plural form of `userNames`.)

Example

```
ectool createGroup "Build Users" --userNames "aallen" "Betty Barker" "cclark"
```

[Back to Top](#)

createUser

Creates a new *local* user.

Note: This API does not apply to non-local users.

User or Group Lists

The commands `createUser` and `modifyUser` can have an optional argument called `groupNames`. The commands

`createGroup` and `modifyGroup` can have an optional argument named `userNames`. In each case, the optional argument is followed by a list of groups or names.

Using ectool, your command string would be:

```
ectool createGroup "New Group Name" --userNames "A Adams" "B Barker"
```

To make this call via the Perl API, create a list of names and then pass a reference to the list as an optional parameter.

Note: The name of the optional parameter is singular, "userName" or "userGroup," not the plural form used by ectool.

Here is an example using the Perl API:

```
# Run the procedure - pass a reference to the list of names
$xPath = $cmdr->createGroup("New Group Name", {
    "userName" => ['A Adams', 'B Burns'] });
```

You must specify a userName.

Arguments	Descriptions
email	The new user's email address.
fullUserName	The user's full name - not his or her nickname.
groupNames	<group1 group2> Any group name containing spaces must be enclosed in double-quotes.
password	The new user's password.
userName	This could be the user's full name, but more commonly it is the shortened name, first initial and last name, or nickname used for email.

Positional arguments

userName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->createUser(<userName>, {<optionals>});

Example

```
$cmdr->createUser("aallen", {fullUserName => "Albert Allen"});
```

ectool

syntax: ectool createUser <userName> ...

Examples

```
ectool createUser "aallen" --fullUserName "Albert Allen"
```

```
ectool createUser "Betty Barker"
```

[Back to Top](#)

deleteGroup

Deletes a local group.

You must specify a `groupName`.

Arguments	Descriptions
<code>groupName</code>	The name of the group you want to delete.

Positional arguments

`groupName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteGroup(<groupName>);`

Example

```
$cmdr->deleteGroup("Build Users");
```

ectool

syntax: `ectool deleteGroup <groupName>`

Example

```
ectool deleteGroup "Build Users"
```

[Back to Top](#)

deleteUser

Deletes a local user.

You must specify the `userName`.

Arguments	Descriptions
userName	The name of the user you want to delete.

Positional arguments

userName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteUser(<userName>);

Example

```
$cmdr->deleteUser("Betty Barker");
```

ectool

syntax: ectool deleteUser <userName>

Example

```
ectool deleteUser "Betty Barker"
```

[Back to Top](#)

getGroup

Retrieves a group by its name.

You must specify the groupName.

Arguments	Descriptions
groupName	The name of the group to retrieve.
providerName	Using this option allows you to search only the specified provider for group information. (LDAP or Active Directory)

Positional arguments

groupName

Response

One [group](#) element.

ec-perl

syntax: \$cmdr->getGroup(<groupName>, {<optionals>});

Example

```
$cmdr->getGroup("myGroup", {providerName => "LDAP"});
```

ectool

syntax: ectool getGroup <groupName> ...

Example

```
ectool getGroup myGroup --providerName LDAP
```

[Back to Top](#)

getGroups

Retrieves all groups.

Arguments	Descriptions
filter	A string used to filter the returned groups by their names.
includeAll	<i><Boolean flag - 0 1 true false></i> When enabled, this argument returns ALL matching groups, including LDAP or non-LDAP groups that may or may not be in the Commander database already. A group is added to the Commander database when a user [who is a member of that group] logs in to Commander for the first time.
maximum	Specifies the maximum number of groups you want to see.

Positional arguments

None

Response

Zero or more [group](#) elements, each containing summary information only.

ec-perl

syntax: \$cmdr->getGroups({ <optionals>});

Example

```
$cmdr->getGroups({filter => " dev*", maximum => 3,});
```

ectool

syntax: ectool getGroups ...

Example

```
ectool getGroups --filter dev* --maximum 3
```

[Back to Top](#)

getUser

Retrieves a user by name.

You must specify the `userName`.

Arguments	Descriptions
<code>providerName</code>	The name of the directory provider. If specified, this option limits the search to the specified directory provider.
<code>userName</code>	The name of the user.

Positional arguments

`userName`

Response

One `user` element.

ec-perl

syntax: `$cmdr->getUser(<userName>, {<optionals>});`

Example

```
$cmdr->getUser("Betty Barker");
```

ectool

syntax: `ectool getUser <userName> ...`

Example

```
ectool getUser "Betty Barker"
```

[Back to Top](#)

getUsers

Retrieves users. By default, this command returns users who have been added to the Commander database, which means they have logged in previously.

Note: When calling `getUsers`, the default limit is 100 user records. Use the `maximum` option to specify a larger number, but this may inhibit performance, or you could define a search pattern to filter your search and conduct multiple queries.

Arguments	Descriptions
<code>filter</code>	<i><filter pattern></i> Supply a filter pattern to match user names. The filter is not case sensitive and can include the "*" wildcard character.
<code>includeAll</code>	<i><Boolean flag - 0 1 true false></i> When enabled, this argument returns ALL matching groups, including LDAP or non-LDAP groups that may or may not be in the Commander database. A group is added to the Commander database when a user who is a member of that group logs in to Commander for the first time.
<code>maximum</code>	<i><number of users></i> Specify a larger number of user records to retrieve. The default limit is 100 user records.

Positional arguments

None

Response

Zero or more [user](#) elements with summary information only.

ec-perl

syntax: `$cmdr->getUsers({<optionals>});`

Examples

```
$cmdr->getUsers();
```

```
$cmdr->getUsers({filter => '*Betty*', maximum => 25});
```

ectool

syntax: `ectool getUsers ...`

Examples

```
ectool getUsers
```

```
ectool getUsers --filter *Betty* --maximum 25
```

[Back to Top](#)

login

Logs into the server and saves the session ID for subsequent ectool use. The user name provided determines the permissions for commands that can be run during the session.

You must specify the `userName` and `password`.

Arguments	Descriptions
password	The password for the user who is "logging in".
userName	The name of a user who has login privileges.

Positional arguments

userName, password

Response

One `session` element containing the session ID.

ec-perl

syntax: \$cmdr->login(<userName>, <password>);

Example

```
$cmdr->login("Ellen Ernst", "ee123");
```

ectool

syntax: ectool login <userName> <password>

Note: ectool will prompt for the password if not supplied.

Example

```
ectool --server EAVMXP login "Ellen Ernst" "ee123"
```

[Back to Top](#)

logout

Logs out of the client session.

Arguments	Descriptions
None	

Positional arguments

None

Response

None or a status OK message.

ec-perl*Example*

```
$cmdr->logout();
```

ectool*Example*

```
ectool logout
```

[Back to Top](#)

modifyGroup

Modifies an existing group.

You must specify `groupName`.

Arguments	Descriptions
<code>groupName</code>	The name of the group to modify.
<code>migrateSettings</code>	<i><targetGroupName></i> Use this argument to specify the new name to which the settings need to be moved.
<code>newName</code>	Supply any name of your choice to rename the group.
<code>removeAllUsers</code>	<i><Boolean flag - 0 1 true false></i>
<code>userNames</code>	<code>user1 [user2...]</code> Provide a complete list of names for the group. These names will replace existing names in the group. Any name with spaces must be enclosed in double-quotes.

Positional arguments

`groupName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->modifyGroup(<groupName>, {...});`

Examples

```
$cmdr->modifyGroup("Build Users", {userName => "dduncan"});
```

```
$cmdr->modifyGroup("Build Users", {userName => ["dduncan", "jack"]});
```


ectool

syntax:ectool modifyGroup <groupName> ...

Examples

```
ectool modifyGroup "Build Users" --userNames dduncan
```

```
ectool modifyGroup "Build Users" --userNames dduncan jack
```

[Back to Top](#)

modifyUser

Modifies an existing *local* user.

Note: This API does *not* apply to non-local users.

User or Group Lists

The commands `createUser` and `modifyUser` can have an optional argument called `groupNames`.

The commands `createGroup` and `modifyGroup` can have an optional argument named `userNames`.

In each case, the optional argument is followed by a list of groups or names.

Using `ectool`, your command string would be:

```
ectool createGroup "New Group Name" --userNames "A Adams" "B Barker"
```

To make this call via the Perl API, create a list of names and then pass a reference to the list as an optional parameter.

Note: The name of the optional parameter is singular, "userName" or "userGroup," not the plural form used by `ectool`.

Here is an example using the Perl API:

```
# Run the procedure - pass a reference to the list of names
$XPath = $cmdr->createGroup("New Group Name", {
    "userName" => ['A Adams', 'B Burns'] });
```

You must specify a `userName`.

Arguments	Descriptions
email	The user's email address.
fullUserName	The user's full name. For example, "John Smith".
groupNames	<i>group1</i> [<i>group2</i> ...] Assigns the user to one or more groups and removes the user from any groups not included in the list.
migrateSettings	<i><targetUserName></i> Use this option to specify the new name to which the settings need to be moved.

Arguments	Descriptions
<code>newName</code>	The user's new name (for example, if changing an existing user's surname).
<code>password</code>	Supply a new password to set for the user.
<code>removeFromAllGroups</code>	<i><Boolean flag - 0 1 true false></i> If set to 1, this user will be removed from all groups.
<code>sessionPassword</code>	If changing the user's password, you must supply the password used in the "login" command also.
<code>userName</code>	The name used by the user to login and/or receive email. For example, "jsmith".

Positional arguments

`userName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->modifyUser(<userName>, {<optionals>});`

Example

```
$cmdr->modifyUser("Betty Barker", {email => "bbarker@abc.com"});
```

ectool

syntax: `ectool modifyUser <userName> ...`

Example

```
ectool modifyUser "Betty Barker" --email "bbarker@abc.com"
```

[Back to Top](#)

removeUsersFromGroup

Removes one or more users from a particular group.

You must specify a `groupName` and one or more user names.

Arguments	Descriptions
groupName	The name of the group from which to remove users.
userNames	The list of users to remove from the group.

Positional arguments

groupName, userNames

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->removeUsersFromGroup(<groupName>, {<optionals>});

Example

```
$cmdr->removeUsersFromGroup("Developers", {userName => ["John", "Jim", "Joey"]});
```

ectool

syntax: ectool removeUsersFromGroup <groupName> <userNames> ...

Example

```
ectool removeUsersFromGroup Developers --userNames John Jim Joey
```

[Back to Top](#)

API Commands - Workflow Management

```
completeWorkflow
deleteWorkflow
getState
getStates
getTransition
getTransitions
getWorkflow
getWorkflows
runWorkflow
transitionWorkflow
```

completeWorkflow

Marks a workflow as completed. When completed, transitions are no longer evaluated.

You must specify `projectName` and `workflowName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project.
<code>workflowName</code>	The name of the workflow.

Positional arguments

`projectName`, `workflowName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->completeWorkflow (<projectName>, <workflowName>{...});`

Example

```
$cmdr->completeWorkflow ("projectA", "workflow_26_201010121647");
```

ectool

syntax: `ectool completeWorkflow <projectName> <workflowName>`

Example

```
ectool completeWorkflow projectA workflow_26_201010121647
```

[Back to Top](#)

deleteWorkflow

Deletes a workflow, including all states and transitions.

You must specify a `projectName` and a `workflowName`.

Arguments	Descriptions
<code>deleteProcesses</code>	<Boolean flag - 0 1 true false>
<code>projectName</code>	The name of the project containing the workflow to delete.
<code>workflowName</code>	The name of the workflow.

Positional arguments

`projectName`, `workflowName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->deleteWorkflow (<projectName>, <workflowName>);`

Example

```
$cmdr->deleteWorkflow ("projectA", "workflow_26_201010121647");
```

ectool

syntax: `ectool deleteWorkflow <projectName> <workflowName> ...`

Example

```
ectool deleteWorkflow projectA workflow_26_201010121647
```

[Back to Top](#)

getState

Finds a state by name.

You must specify `projectName`, `workflowName`, and `stateName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the state.

Arguments	Descriptions
stateName	The name of the state.
workflowName	The name of the workflow.

Positional arguments

projectName, workflowName, stateName

Response

One [state](#) element.

ec-perl

syntax: \$cmdr->getState (<projectName>, <workflowName>, <stateName>);

Example

```
$cmdr->getState ("projectA", "workflow_26_201010121647", "build");
```

ectool

syntax: ectool getState <projectName> <workflowName> <stateName>

Example

```
ectool getState projectA workflow_26_201010121647 build
```

[Back to Top](#)

getStates

Retrieves all states in a workflow.

You must specify projectName and workflowName.

Arguments	Descriptions
projectName	The name of the project containing the state.
workflowName	The name of the workflow.

Positional arguments

projectName, workflowName

Response

One or more [state](#) elements.

ec-perl

syntax: \$cmdr->getStates (projectName>, <workflowName>);

Example

```
$cmdr->getStates ("projectA", "workflow_26_201010121647");
```

ectool

syntax: ectool getStates <projectName> <workflowName>

Example

```
ectool getStates projectA workflow_26_201010121647
```

[Back to Top](#)

getTransition

Finds a transition by name.

You must specify projectName, workflowName, stateName, and transitionName.

Arguments	Descriptions
projectName	The name of the project containing the transition.
stateName	The name of the state.
transitionName	The name of the transition.
workflowName	The name of the workflow.

Positional arguments

```
projectName, workflowName, stateName, transitionName
```

Response

One [transition](#) element.

ec-perl

syntax: \$cmdr->getTransition (projectName>, <workflowName>, <stateName>, <transitionName>);

Example

```
$cmdr->getTransition ("projectA", "workflow_26_201010121647", "build", "build2test");
```

ectool

syntax: ectool getTransition <projectName> <workflowName> <stateName>
<transitionName>

Example

```
ectool getTransition projectA workflow_26_201010121647 build build2test
```

[Back to Top](#)

getTransitions

Retrieves all transitions in a workflow.

You must specify `projectName`, `workflowName`, and `stateName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the transition.
<code>stateName</code>	The name of the state.
<code>targetState</code>	The target state for the transition definition.
<code>workflowName</code>	The name of the workflow.

Positional arguments

`projectName`, `workflowName`, `stateName`

Response

One or more [transition](#) elements.

ec-perl

syntax: \$cmdr->getTransitions (<projectName>, <workflowName>, <stateName>);

Example

```
$cmdr->getTransitions ("projectA", "workflow_26_201010121647", "build");
```

ectool

syntax: ectool getTransitions <projectName> <workflowName> <stateName>

Example

```
ectool getTransitions projectA workflow_26_201010121647 build
```

[Back to Top](#)

getWorkflow

Finds a workflow by name.

You must specify a `projectName` and `workflowName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the workflow.
<code>workflowName</code>	The name of the workflow.

Positional arguments

`projectName`, `workflowName`

Response

One `workflow` element.

ec-perl

syntax: `$cmdr->getWorkflow (<projectName>, <workflowName>);`

Example

```
$cmdr->getWorkflow ("projectA", "BTD");
```

ectool

syntax: `ectool getWorkflow <projectName> <workflowName>`

Example

```
ectool getWorkflow projectA BTD
```

[Back to Top](#)

getWorkflows

Retrieves all workflow instances in a project.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the workflows.

Positional arguments

`projectName`

Response

Zero or more [workflow](#) elements.

ec-perl

syntax: `$cmdr->getWorkflows (<projectName>);`

Example

```
$cmdr->getWorkflows ("projectA");
```

ectool

syntax: `ectool getWorkflows <projectName>`

Example

```
ectool getWorkflows projectA
```

[Back to Top](#)

runWorkflow

Runs the specified workflow definition and returns the workflow name.

You must specify the `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the workflow starting state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the starting state.
<code>projectName</code>	The name of the project containing the workflow definition.
<code>startingState</code>	The initial state of the workflow.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

`projectName, workflowDefinitionName`

Response

The workflow name is returned.

ec-perl

syntax: `$cmdr->runWorkflow (<projectName>, <workflowDefinitionName>, {<optionals>});`

Example

```
$cmdr->runWorkflow ("projectA", "BTD", {startingState => "build"});
```

ectool

syntax: ectool runWorkflow <projectName> <workflowDefinitionName> ...

Example

```
ectool runWorkflow projectA BTD --startingState build
```

[Back to Top](#)

transitionWorkflow

Manually transition from the active workflow state.

You must specify `projectName`, `workflowName`, `stateName`, and `transitionName`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the transition's target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state.
<code>projectName</code>	The name of the project containing the workflow to transition.
<code>stateName</code>	The name of the state.
<code>transitionName</code>	The name of the transition.
<code>workflowName</code>	The name of the workflow to transition.

Positional arguments

```
projectName, workflowName, stateName, transitionName
```

Response

None or status OK message.

ec-perl

syntax: \$cmdr->transitionWorkflow (<projectName>, <workflowName>, <stateName>, <transitionName>, {<optionals>});

Example

```
$cmdr->transitionWorkflow ("projectA", "workflow_26_201010121647", "build", "build2  
test");
```

ectool

syntax: ectool transitionWorkflow <projectName> <workflowName> <stateName>
<transitionName> ...

Example

```
ectool transitionWorkflow projectA workflow_26_201010121647 build build2test
```

[Back to Top](#)

API Commands - Workflow Definition Management

<code>createStateDefinition</code>	<code>getTransitionDefinitions</code>
<code>createTransitionDefinition</code>	<code>getWorkflowDefinition</code>
<code>createWorkflowDefinition</code>	<code>getWorkflowDefinitions</code>
<code>deleteStateDefinition</code>	<code>modifyStateDefinition</code>
<code>deleteTransitionDefinition</code>	<code>modifyTransitionDefinition</code>
<code>deleteWorkflowDefinition</code>	<code>modifyWorkflowDefinition</code>
<code>getStateDefinition</code>	<code>moveStateDefinition</code>
<code>getStateDefinitions</code>	<code>moveTransitionDefinition</code>
<code>getTransitionDefinition</code>	

createStateDefinition

Creates a new state definition for a workflow definition. Optionally, a state may launch either a procedure or a sub-workflow as its "process" when the state is entered.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the process. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the process. For more information about parameters, click here .
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>projectName</code>	The name of the project.
<code>startable</code>	<Boolean flag - 0 1 true false> "True" means this state definition can be the initial state of an instantiated workflow.
<code>stateDefinitionName</code>	Choose any unique name of your choice for the state definition. This name must be unique within the workflow definition.

Arguments	Descriptions
subprocedure	Name of the procedure launched when the state is entered. Also requires subproject
subproject	Name of the project containing the procedure or workflow launched when the state is entered.
substartingState	Name of the starting state for the workflow launched when the state is entered. Also requires subproject and subworkflowDefinition
subworkflowDefinition	Name of the workflow definition launched when the state is entered. Also requires subproject
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response

One [stateDefinition](#) element.

ec-perl

syntax: \$cmdr->createStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, {<optionals>});

Example

```
$cmdr->createStateDefinition ("ProjectA", "BTD", "build", {startable => 1,
    subproject => "product",
    subprocedure => "Master",
    description => "free text"});
```

ectool

syntax: ectool createStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...

Example

```
ectool createStateDefinition ProjectA BTD build --startable 1 --subproject product
--subprocedure Master --description "free text"
```

[Back to Top](#)

createTransitionDefinition

Creates a new transition definition for workflow definition.

You must specify `projectName`, `workflowDefinitionName`, `stateDefinitionName`, `transitionDefinitionName`, and `targetState`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state. For more information about parameters, click here .
<code>condition</code>	A fixed text or text embedding property references that are evaluated into a logical TRUE or FALSE. An empty string, a "0" or "false" is interpreted as FALSE. Any other result string is interpreted as TRUE. This field is ignored by the server if <code>trigger</code> is set to manual.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>projectName</code>	The name of the project.
<code>stateDefinitionName</code>	The name of the state definition.
<code>targetState</code>	Target state for the transition definition.
<code>transitionDefinitionName</code>	Choose any unique name of your choice for the transition definition. This name must be unique within the state definition.
<code>trigger</code>	Possible values are: <code>onEnter</code> <code>onStart</code> <code>onCompletion</code> <code>manual</code>
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

`projectName`, `workflowDefinitionName`, `stateDefinitionName`,
`transitionDefinitionName`, `targetState`

Response

One `transitionDefinition` element.

ec-perl

syntax: `$cmdr->createTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>, <targetState>, {<optionals>});`

Example

```
$cmdr->createTransitionDefinition ("ProjectA", "BTD", "build", "build2test", "test",
    {trigger => "manual", description => "free text"});
```

ectool

syntax: `ectool createTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> <targetState> ...`

Example

```
ectool createTransitionDefinition ProjectA BTD build build2test test --trigger manual
--description "free text"
```

[Back to Top](#)

createWorkflowDefinition

Creates a new workflow definition for a project.

You must supply a `projectName` and a `workflowDefinitionName`.

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
projectName	The name of the project containing the workflow.
workflowDefinitionName	Choose any unique name of your choice for the workflow definition. This name must be unique within the project.
workflowNameTemplate	The name of the workflow template.

Positional arguments

projectName, workflowDefinitionName

Response

One `workflowDefinition` element.

ec-perl

syntax: `$cmdr->createWorkflowDefinition (projectName>, <workflowDefinitionName>,
{<optionals>});`

Example

```
$cmdr->createWorkflowDefinition ("projectA", "BTD", {description => "free text"});
```

ectool

syntax: `ectool createWorkflowDefinition <projectName> <workflowDefinitionName> ...`

Example

```
ectool createWorkflowDefinition projectA BTD --description "free text"
```

[Back to Top](#)

deleteStateDefinition

Deletes a state definition.

You must specify a `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
projectName	The name of the project containing the state definition.
workflowDefinitionName	The name of the workflow definition.
stateDefinitionName	The name of the state definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response

None or status OK message.

ec-perl

syntax: `$cmdr->deleteStateDefinition (<projectName>, <workflowDefinitionName>,
<stateDefinitionName>);`

Example

```
$cmdr->deleteStateDefinition ("projectA", "BTD", "build");
```

ectool

syntax: ectool deleteStateDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName>

Example

```
ectool deleteStateDefinition projectA BTD build
```

[Back to Top](#)

deleteTransitionDefinition

Deletes a transition definition.

You must specify a projectName, workflowDefinitionName, stateDefinitionName, and transitionDefinitionName.

Arguments	Descriptions
projectName	The name of the project containing the transition definition.
stateDefinitionName	The name of the state definition.
transitionDefinitionName	The name of the transition definition.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

Response

None or status OK message.

ec-perl

syntax: \$cmdr->deleteTransitionDefinition (<projectName>, <workflowDefinitionName>,
<stateDefinitionName>, <transitionDefinitionName>);

Example

```
$cmdr->deleteTransitionDefinition ("projectA", "BTD", "build", "build2test");
```

ectool

syntax: ectool deleteTransitionDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName> <transitionDefinitionName>

Example

```
ectool deleteTransitionDefinition projectA BTD build build2test
```

[Back to Top](#)

deleteWorkflowDefinition

Deletes a workflow definition, including all state and transition definitions.

You must specify a `projectName` and a `workflowDefinitionName`

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the workflow definition to delete.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

```
projectName,workflowDefinitionName
```

Response

None or status OK message.

ec-perl

syntax: `$cmdr->deleteWorkflowDefinition (<projectName>, <workflowDefinitionName>);`

Example

```
$cmdr->deleteWorkflowDefinition ("projectA", "BTD");
```

ectool

syntax: `ectool deleteWorkflowDefinition <projectName> <workflowDefinitionName>`

Example

```
ectool deleteWorkflowDefinition projectA BTD
```

[Back to Top](#)

getStateDefinition

Finds a state definition by name.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
projectName	The name of the project containing the state definition.
stateDefinitionName	The name of the state definition.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response

One `stateDefinition` element.

ec-perl

syntax: `$cmdr->getStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);`

Example

```
$cmdr->getStateDefinition ("projectA", "BTD", "build");
```

ectool

syntax: `ectool getStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName>`

Example

```
ectool getStateDefinition projectA BTD build
```

[Back to Top](#)

getStateDefinitions

Retrieves all state definitions in a workflow definition.

You must specify `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
includeFormalParameters	<i><Boolean flag - 0 1 true false></i>
projectName	The name of the project containing the state definition.
startableOnly	<i><Boolean flag - 0 1 true false></i>
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName

Response

One or more [stateDefinition](#) elements.

ec-perl

syntax: \$cmdr->getStateDefinitions (<projectName>, <workflowDefinitionName>, {<optionals>});

Example

```
$cmdr->getStateDefinitions ("projectA", "BTD", {startableOnly => 1});
```

ectool

syntax: ectool getStateDefinitions <projectName> <workflowDefinitionName> ...

Example

```
ectool getStateDefinitions projectA BTD --startableOnly 1
```

[Back to Top](#)

getTransitionDefinition

Finds a transition definition by name.

You must specify projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName.

Arguments	Descriptions
projectName	The name of the project containing the transition definition.
stateDefinitionName	The name of the state definition.
transitionDefinitionName	The name of the transition definition.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

Response

One [transitionDefinition](#) element.

ec-perl

syntax: \$cmdr->getTransitionDefinition (<projectName>, <workflowDefinitionName>,
<stateDefinitionName>, <transitionDefinitionName>);

Example

```
$cmdr->getTransitionDefinition ("projectA", "BTD", "build", "build2test");
```

ectool

syntax: ectool getTransitionDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName> <transitionDefinitionName>

Example

```
ectool getTransitionDefinition projectA BTD build build2test
```

[Back to Top](#)

getTransitionDefinitions

Retrieves all transition definitions in a workflow definition.

You must specify `projectName`, `stateDefinitionName`, `workflowDefinitionName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the transition definitions.
<code>stateDefinitionName</code>	The name of the state definition.
<code>targetState</code>	The name of the target state.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

`projectName`, `stateDefinitionName`, `workflowDefinitionName`

Response

Zero or more [transitionDefinition](#) elements.

ec-perl

syntax: \$cmdr->getTransitionDefinitions (<projectName>, <stateDefinitionName>,
<workflowDefinitionName>, {<optionals>});

Example

```
$cmdr->getTransitionDefinitions ("projectA", "build", "BTD");
```

ectool

syntax: `ectool getTransitionDefinitions <projectName> <stateDefinitionName>
<workflowDefinitionName> ...`

Example

```
ectool getTransitionDefinitions projectA build BTD
```

[Back to Top](#)

getWorkflowDefinition

Finds a workflow definition by name.

You must specify a `projectName` and a `workflowDefinitionName`.

Arguments	Descriptions
projectName	The name of the project containing the workflow definition.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName

Response

One workflowDefinition element.

ec-perl

syntax: \$cmdr->getWorkflowDefinition (<projectName>, <workflowDefinitionName>);

Example

```
$cmdr->getWorkflowDefinition ("projectA", "BTD");
```

ectool

syntax: ectool getWorkflowDefinition <projectName> <workflowDefinitionName>

Example

```
ectool getWorkflowDefinition projectA BTD
```

[Back to Top](#)

getWorkflowDefinitions

Retrieves all workflow definitions in a project.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the workflow definitions.

Positional arguments

`projectName`

Response

Zero or more `workflowDefinition` elements.

ec-perl

syntax: `$cmdr->getWorkflowDefinitions (<projectName>);`

Example

```
$cmdr->getWorkflowDefinitions ("projectA");
```

ectool

syntax: `ectool getWorkflowDefinitions <projectName>`

Example

```
ectool getWorkflowDefinitions projectA
```

[Back to Top](#)

modifyStateDefinition

Modifies an existing state definition.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
<code>actualParameter</code>	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.
<code>clearActualParameters</code>	<i><Boolean flag - 0 1 true false></i>

Arguments	Descriptions
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
newName	The new name of your choice for the state definition.
projectName	The name of the project containing the state definition.
startable	<i><Boolean flag - 0 1 true false></i>
stateDefinitionName	The name of the state definition to modify.
subprocedure	The name of the procedure launched when the state is entered. Also requires subproject
subproject	The name of the project containing the procedure or workflow launched when the state is entered.
substartingState	The name of the workflow starting state that is launched when the state is entered. Also requires subproject and subworkflowDefinition
subworkflowDefinition	The name of the workflow definition launched when the state is entered. Also requires subproject
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response

One [stateDefinition](#) element.

ec-perl

syntax: \$cmdr->modifyStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);

Example

```
$cmdr->modifyStateDefinition ("projectA", "BTD", "build",
    {startable => 1,
      subproject => "factory",
      subprocedure => "Master",
      description => "sample text"});
```

ectool

syntax: ectool modifyStateDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName> ...

Example

```
ectool modifyStateDefinition projectA BTD build --startable 1 --subproject factory
--subprocedure Master --description "sample text"
```

[Back to Top](#)

modifyTransitionDefinition

Modifies an existing transition definition.

You must specify `projectName`, `workflowDefinitionName`, `stateDefinitionName`, and `transitionDefinitionName`.

Arguments	Descriptions
<code>actualParameter</code>	Specifies the values to pass as parameters to the target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state.
<code>clearActualParameters</code>	<Boolean flag - 0 1 true false>
<code>condition</code>	A fixed text or text embedded property references that are evaluated into a logical "true" or "false". An empty string, a "0" or "false" is interpreted as "false". Any other result string is interpreted as "true". This field is ignored by the server if <code>trigger</code> is set to manual.
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>

Arguments	Descriptions
newName	A new name of your choice for the transition definition--must be a unique name within the workflow.
projectName	The name of the project containing the transition definition to modify.
stateDefinitionName	The name of the state definition.
targetState	The target state for the transition definition.
transitionDefinitionName	The name of the transition definition to modify.
trigger	Possible values are: onEnter onStart onCompletion manual
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName,
transitionDefinitionName

Response

One [transitionDefinition](#) element.

ec-perl

syntax: \$cmdr->modifyTransitionDefinition (<projectName>, <workflowDefinitionName>,
<stateDefinitionName>, <transitionDefinitionName>, {<optionals>});

Example

```
$cmdr->modifyTransitionDefinition ("projectA", "BTD", "build", "build2test",
    {targetState => "deploy",
      trigger => "onCompletion",
      description => "bypass all tests"});
```

ectool

syntax: ectool modifyTransitionDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName> <transitionDefinitionName> ...

Example

```
ectool modifyTransitionDefinition projectA BTD build build2test
--targetState deploy
--trigger onCompletion
--description "bypass all tests"
```

[Back to Top](#)

modifyWorkflowDefinition

Modifies an existing workflow definition.

You must specify `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
<code>description</code>	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<code>newName</code>	The new name of your choice for the workflow definition--must be a unique name within the workflow.
<code>projectName</code>	The name of the project containing the workflow definition to modify.
<code>workflowDefinitionName</code>	The name of the workflow definition to modify.
<code>workflowNameTemplate</code>	The template used to determine default names for workflows launched from a workflow definition.

Positional arguments

`projectName`, `workflowDefinitionName`

Response

One `workflowDefinition` element.

ec-perl

syntax: `$cmdr->modifyWorkflowDefinition (<projectName>, <workflowDefinitionName>, {<optionals>});`

Example

```
$cmdr->modifyWorkflowDefinition ("projectA", "BTD",  
    {newName => "BuildTestDeploy",  
    description => "changed name"});
```

ectool

syntax: `ectool modifyWorkflowDefinition <projectName> <workflowDefinitionName> ...`

Example

```
ectool modifyWorkflowDefinition projectA BTD
  --newName "BuildTestDeploy"
  --description "changed name"
```

[Back to Top](#)

moveStateDefinition

Moves a state definition within a workflow definition.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
<code>beforeStateDefinition</code>	Use this option to reorder state definitions in a workflow definition. The state definition (<code>stateDefinitionName</code>) will be moved to a position just before the state definition "named" by this option. If omitted, the state definition is moved to the end of the workflow definition.
<code>projectName</code>	The name of the project containing the state definition.
<code>stateDefinitionName</code>	The name of the state definition to move.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

`projectName`, `workflowDefinitionName`, `stateDefinitionName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->moveStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, {<optionals>});`

Example

```
$cmdr->moveStateDefinition ("projectA", "BTD", "deploy",
  {beforeStateDefinition => "test"});
```

ectool

syntax: `ectool moveStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...`

Example

```
ectool moveStateDefinition projectA BTD deploy --beforeStateDefinition test
```

[Back to Top](#)

moveTransitionDefinition

Moves a transition definition within a workflow definition.

You must specify `projectName`, `workflowDefinitionName`, `stateDefinitionName`, and `transitionDefinitionName`.

Arguments	Descriptions
<code>beforeTransitionDefinition</code>	Use this option to move a transition definition in a workflow definition. The transition definition is moved to a position just before the transition definition named by this option. If omitted, the transition definition is moved to the end of the workflow definition.
<code>projectName</code>	The name of the project containing the transition definition.
<code>stateDefinitionName</code>	The name of the state definition.
<code>transitionDefinitionName</code>	The name of the transition definition to move.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Positional arguments

```
projectName, workflowDefinitionName, stateDefinitionName,  
transitionDefinitionName
```

Response

None or status OK message.

ec-perl

syntax: `$cmdr->moveTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>, {<optionals>});`

Example

```
$cmdr->moveTransitionDefinition ("projectA", "BTD", "Build", "in",  
    {beforeTransitionDefinition => "out"});
```

ectool

syntax: `ectool moveTransitionDefinition <projectName> <workflowDefinitionName>
 <stateDefinitionName> <transitionDefinitionName> ...`

Example

```
ectool moveTransitionDefinition projectA BTD Build in--beforeTransitionDefinition o  
ut
```

[Back to Top](#)

API Commands - Workspace Management

```
createWorkspace  
deleteWorkspace  
getWorkspace  
getWorkspaces  
modifyWorkspace
```

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`:

<code>agentDrivePath</code>	<code>agentUncPath</code>	Result from running a step in "job123" that uses this workspace
N:\	\\server\share	The agent maps \\server\share to drive n: and runs the step in n:\job123.
N:\sub1	\\server\share\dir1\sub1	The agent maps \\server\share\dir1 to drive n: and runs the step in n:\sub1\job123.
N:\sub1	\\server\share\dir1	Invalid! No mapping can be deduced from this pair of values.
C:\ws	C:\ws	A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.
C:\ws		Same as if <code>agentUncPath</code> were set identical to <code>agentDrivePath</code> .

You must specify a `workspaceName`.

Arguments	Descriptions
agentDrivePath	Drive-letter-based path used by Windows agents to access the workspace in steps.
agentUncPath	UNC path used by Windows Commander Web servers to access the workspace. The agent uses <code>agentUncPath</code> and <code>agentDrivePath</code> to compute the drive mapping needed for making <code>agentDrivePath</code> valid in the step.
agentUnixPath	UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.
credentialName	Credential to use when connecting to a network location. <code>credentialName</code> can be one of two forms: relative (for example, " <code>cred1</code> ") - the credential is assumed to be in the project that contains the request target object absolute (for example, " <code>/projects/BuildProject/credentials/cred1</code> ") - the credential can be from any specified project, regardless of the target object's project.
description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
local	<Boolean flag - 0 1 true false> Set to "true", the workspace is local.
workspaceDisabled	<Boolean flag - 0 1 true false> Set to "true", the workspace is disabled.
workspaceName	Any name you choose to name this workspace.
zoneName	The name of the zone where this workspace resides.

Positional arguments

`workspaceName`

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->createWorkspace(<workspaceName>, {<optionals>});

Example

```
$cmdr->createWorkspace('test', {agentDrivePath => 'c:/workspace',  
    agentUncPath => 'c:/workspace',  
    agentUnixPath => '/mnt/server/workspace'});
```

ectool

syntax: ectool createWorkspace <workspaceName> ...

Example

```
ectool createWorkspace test --agentDrivePath c:/workspace --agentUncPath  
c:/workspace --agentUnixPath '/mnt/server/workspace'
```

[Back to Top](#)

deleteWorkspace

Deletes a workspace.

You must specify the workspaceName.

Arguments	Descriptions
workspaceName	The name of the workspace to delete.

Positional arguments

workspaceName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->deleteWorkspace(<workspaceName>);

Example

```
$cmdr->deleteWorkspace("test");
```

ectool

syntax: ectool deleteWorkspace <workspaceName>

Example

```
ectool deleteWorkspace test
```

[Back to Top](#)

getWorkspace

Retrieves a workspace by name.

You must specify the `workspaceName`.

Arguments	Descriptions
<code>workspaceName</code>	The name of the workspace to retrieve.

Positional arguments

`workspaceName`

Response

One [workspace](#) element.

ec-perl

syntax: `$cmdr->getWorkspace(<workspaceName>);`

Example

```
$cmdr->getWorkspace("test");
```

ectool

syntax: `ectool getWorkspace <workspaceName>`

Example

```
ectool getWorkspace test
```

[Back to Top](#)

getWorkspaces

Retrieves all workspaces.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [workspace](#) elements.

ec-perl

syntax: `$cmdr->getWorkspaces()` ;

Example

```
$cmdr->getWorkspaces();
```

ectool

syntax: `ectool getWorkspaces`

Example

```
ectool getWorkspaces
```

[Back to Top](#)

modifyWorkspace

Modifies an existing workspace.

A workspace definition consists of three paths to access the workspace in various ways:

`agentDrivePath`

`agentUncPath` - The agent uses `agentUncPath` and `agentDrivePath` to compute the drive mapping needed to make `agentDrivePath` valid in the step (see examples below).

`agentUnixPath`

Examples for `agentDrivePath` and `agentUncPath`:

agentDrivePath	agentUncPath	Result from running a step in "job123" that uses this workspace
N:\	\\server\share	The agent maps \\server\share to drive n: and runs the step in n:\job123.
N:\sub1	\\server\share\dir1\sub1	The agent maps \\server\share\dir1 to drive n: and runs the step in n:\sub1\job123.
N:\sub1	\\server\share\dir1	Invalid! No mapping can be deduced from this pair of values.
C:\ws	C:\ws	A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.
C:\ws		Same as if <code>agentUncPath</code> were set identical to <code>agentDrivePath</code> .

You must specify a `workspaceName`.

Arguments	Descriptions
<code>agentDrivePath</code>	Drive-letter-based path used by Windows agents to access the workspace in steps.
<code>agentUncPath</code>	UNC path used by Windows Commander web servers to access the workspace. The agent uses <code>agentUncPath</code> and <code>agentDrivePath</code> to compute the drive mapping needed for making <code>agentDrivePath</code> valid in the step.
<code>agentUnixPath</code>	UNIX path used by UNIX agents and Linux Commander web servers to access the workspace.
<code>credentialName</code>	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>description</code>	<p>A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
<code>local</code>	<Boolean flag - 0 1 true false> Set to "true", the workspace is local.
<code>newName</code>	Supply any name of your choice to rename the workspace.
<code>workspaceName</code>	The name of the workspace to modify.
<code>workspaceDisabled</code>	<Boolean flag - 0 1 true false> Set to "true", the workspace is disabled.
<code>zoneName</code>	The name of the zone where this workspace resides.

Positional arguments

workspaceName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->modifyWorkspace(<workspaceName>, {<optionals>});

Example

```
$cmdr->modifyWorkspace("test", {description => "my test workspace"});
```

ectool

syntax: ectool modifyWorkspace <workspaceName> ...

Example

```
ectool modifyWorkspace test --description "my test workspace"
```

[Back to Top](#)

API Commands - Miscellaneous Management

[changeOwner](#)
[clone](#)
[countObjects](#)
[deleteObjects](#)
[export](#)
[findObjects](#)
[getObjects](#)
[import](#)

changeOwner

Changes the owner of an object.

You must specify an object name.

Note: The modify privilege on the "admin" system ACL is required to change an object's owner.

For email notifiers, the owner can be changed if the current user has sufficient privileges to have deleted the object and recreated it.

Arguments	Descriptions
credentialName	<p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
configName	The name of the email configuration.
groupName	The full name of a group. For Active Directory and LDAP, this is a full DN.
newOwnerName	The name of the new owner for this object. Defaults to the current user.
notifierName	The name of the email notifier.

Arguments	Descriptions
pluginName	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure - may be a path to the procedure. Also requires projectName
projectName	The name of the project - may be a path. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
resourceName	The name of the resource.
scheduleName	The name of the schedule - may be a path to the schedule. Also requires projectName
stateDefinitionName	The name of the state definition.
stepName	The name of the step - may be a path to the step. Also requires projectName and procedureName
transitionDefinitionName	The name of the transition definition.
userName	The full name of the user. For Active Directory and LDAP, the name may be user@domain.
workflowDefinitionName	The name of the workflow definition.
workspaceName	The name of the workspace.

Positional arguments

None

Response

Returns the modified object.

ec-perl

syntax: \$cmdr->changeOwner ({...});

Example

```
$cmdr->changeOwner ("projectName"=>"Sample Project");
```


ectool

syntax: ectool changeOwner ...

Example

```
ectool changeOwner --projectName "Sample Project"
```

[Back to Top](#)

clone

Makes a copy of an existing ElectricCommander object. For example: credential, directory provider, email configuration, email notifier, project, procedure, property sheet, resource, resource pool, schedule, state definition, step, transition definition, workflow definition, and workspace.

Note: You cannot clone parameters.

IMPORTANT:

To find the entity you want to clone, you must specify the following arguments:

- A new name for the cloned object (cloneName)
- Locator arguments

For example, if you want to clone a project, you must specify the name of the project that you want to clone.

Arguments	Descriptions
Naming	
cloneName	<p>The cloneName specifies the path to the new object, possibly in an alternate location.</p> <p>If no container path is specified, the new object is created inside the same container as the original.</p> <p>If no name is specified, the server will generate a name.</p>
Locators	
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact container of the property sheet which owns the property.

Arguments	Descriptions
artifactVersionName	The name of the artifactVersion container of the property sheet which owns the property..
componentName	The name of the component container of the property sheet which owns the property.
configName	The name of the emailConfig container that owns the property.
credentialName	<p>The name of the credential container of the property sheet which owns the property.</p> <p>credentialName can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
environmentName	The name of the environment container of the property sheet which owns the property; must be unique among all projects.
environmentTierName	The name of the environment tier container of the property sheet which owns the property.
gatewayName	The name of the gateway container of the property sheet.
groupName	The name of the group container of the property sheet which owns the property.
jobId	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier.
objectId	The object id as returned by FindObjects.
path	The property path that specifies the object.

Arguments	Descriptions
pluginName	The name of the plugin container of the property sheet which owns the property.
procedureName	The name of the procedure you want to clone. Also requires projectName
processName	The name of the process, if the container is a process or process step.
processStepName	The name of the process step, if the container is a process step.
projectName	The name of the project you want to clone.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
providerName	The LDAP or Active Directory provider name.
resourceName	The name of the resource you want to clone.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule you want to clone. Also requires projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step you want to clone. Also requires projectName and procedureName
systemObjectName	System object names include: admin artifacts directory emailConfigs forceAbort licensing log plugins priority projects repositories resources server session workspaces zonesAndGateways
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user where you may need to expand the string.

Arguments	Descriptions
<code>workflowDefinitionName</code>	The name of the workflow definition.
<code>workflowName</code>	The name of the workflow.
<code>workspaceName</code>	The name of the workspace you want to clone.
<code>zoneName</code>	The name of the zone.

Positional arguments

None.

Response

Returns the name of the new cloned object.

Using the `clone` command successfully depends on the context of the locator arguments in your system.

The command works when the arguments are specified correctly.

ec-perl

syntax: `$cmdr->clone ...;`

Examples

```
# Create a copy of a procedure, as though you clicked the "Copy"
# button in the UI.
```

```
$XPath = $cmdr->clone(
{
    projectName => "EC-Examples",
    procedureName => "set Property"
}
);
```

```
# Create a copy of a procedure providing a name for the copy.
```

```
$XPath = $cmdr->clone(
{
    projectName    => "EC-Examples",
    procedureName  => "set Property",
    cloneName      => "set Property 2"
}
);
```

```
# Create a copy of a procedure step.
```

```
$XPath = $cmdr->clone(
{
```

```

        projectName => "EC-Examples",
        procedureName => "set Property",
        cloneName => "set Property 2",
        stepName => 'setProperty'
    }
};

# Copy a step using the path.

$xPath = $cmdr->clone(
    {
        path =>
            '/projects/EC-Examples/procedures/set Property/steps/setProperty'
    }
);

```

ectool

syntax: ectool clone ...

Examples

```

# Create a copy of a procedure, as though you clicked the "Copy"
# button in the UI.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
<response requestId="1" nodeId="192.168.16.238">
    <cloneName>Set Property copy</cloneName>
</response>

# Create a copy of a procedure providing a name for the copy.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
--cloneName 'set Property 2'
<response requestId="1" nodeId="192.168.16.238">
    <cloneName>set Property 2</cloneName>
</response>

# Create a copy of a procedure step.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
--stepName 'setProperty'
<response requestId="1" nodeId="192.168.16.238">
    <cloneName>setProperty copy</cloneName>
</response>

# Create a copy of a procedure step using the full path.

$ ectool clone --path '/projects/EC-Examples/procedures/set Property/steps/setPrope
rty'
<response requestId="1" nodeId="192.168.16.238">
    <cloneName>setProperty copy</cloneName>
</response>

```

[Back to Top](#)

countObjects

This API returns the count of objects specified by the provided filter.

Arguments	Descriptions
filter	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>Each element of the filter list is a hash reference containing one filter criterion. You can specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p> <p>Two types of filters:</p> <p>"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p>"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.</p> <p>Property filter operators are:</p> <pre> between (2 operands) contains (1) equals (1) greaterOrEqual (1) greaterThan (1) in (1) lessOrEqual (1) lessThan (1) like (1) notEqual (1) notLike (1) isNotNull (0) isNull (0) </pre> <p>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.</p> <p>Boolean operators are:</p>

Arguments	Descriptions																										
	not (1 operand) and (2 or more operands) or (2 or more operands)																										
objectType	<p>This argument specifies the type of object to count. Values include:</p> <table> <tr><td>artifact</td><td>project</td></tr> <tr><td>artifactVersion</td><td>property</td></tr> <tr><td>credential</td><td>repository</td></tr> <tr><td>directoryProvider</td><td>resource</td></tr> <tr><td>emailconfig</td><td>resourcePool</td></tr> <tr><td>emailNotifier</td><td>schedule</td></tr> <tr><td>formalParameter</td><td>state</td></tr> <tr><td>job</td><td>stateDefinition</td></tr> <tr><td>jobStep</td><td>transition</td></tr> <tr><td>logEntry</td><td>transitionDefinition</td></tr> <tr><td>plugin</td><td>workflow</td></tr> <tr><td>procedure</td><td>workflowDefinition</td></tr> <tr><td>procedureStep</td><td>workspace</td></tr> </table>	artifact	project	artifactVersion	property	credential	repository	directoryProvider	resource	emailconfig	resourcePool	emailNotifier	schedule	formalParameter	state	job	stateDefinition	jobStep	transition	logEntry	transitionDefinition	plugin	workflow	procedure	workflowDefinition	procedureStep	workspace
artifact	project																										
artifactVersion	property																										
credential	repository																										
directoryProvider	resource																										
emailconfig	resourcePool																										
emailNotifier	schedule																										
formalParameter	state																										
job	stateDefinition																										
jobStep	transition																										
logEntry	transitionDefinition																										
plugin	workflow																										
procedure	workflowDefinition																										
procedureStep	workspace																										

Positional arguments

objectType

Response

Returns the number of filtered objects.

ec-perl

syntax: \$cmdr->countObjects(<objectType>, {<optionals>});

Example

```
use ElectricCommander();
my @artifactNameFilters;
# Create the filter list for filtering on artifact name
push (@artifactNameFilters,
    { "propertyName"=>"artifactName",
      "operator"=>"contains",
      "operand1"=>"groupId:installer-windows",
    });
my $cmdr = new ElectricCommander();
# Perform the countObjects query
my $reference=$cmdr->countObjects("artifactVersion",
    { filter=>
      {operator=>"and",
        filter=>[
          { propertyName=>"modifyTime" ,
            operator=>"greaterOrEqual",# Give me all dates after or equal
            "operand1"=>"2014-03-25T14:48:55.286Z",
          }
        ]
      }
    }
    arbitrary date
```



```
        }
      },
      {
        operator => 'or', # apply 'or' for the filters in the list
        filter => \@artifactNameFilter
      }
    ]
  }
});

my $jobs=$reference->find('//response/count');
print $jobs;
```

ectool

Not supported.

[Back to Top](#)

deleteObjects

This API deletes objects specified by the provided filters.

Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.

You must specify an `objectType` and at least one filter.

Note: Currently, this API supports deleting `artifact`, `artifactVersion`, `job`, `logEntry`, `project`, `repository`, and `workflow`.

Arguments	Descriptions
filter	<p>Specify filters in a space-separated list: <code>filter1 filter2 ...</code> A list of zero or more filter criteria definitions used to define objects to find.</p> <p>Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p> <p>Two types of filters:</p> <p>"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p>"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.</p> <p>Property filter operators are:</p> <pre> between (2 operands) contains (1) equals (1) greaterOrEqual (1) greaterThan (1) in (1) lessOrEqual (1) lessThan (1) like (1) notEqual (1) notLike (1) isNotNull (0) isNull (0) </pre> <p>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) </pre>

Arguments	Descriptions
	or (2 or more operands)
maxIds	<p><id count></p> <p>The maximum number of objects that will be deleted. Defaults to all objects that match the filter.</p>
objectType	<p>This argument specifies the type of object to find.</p> <p>Values include:</p> <p>artifact artifactVersion job logEntry project repository workflow</p>
sorts	<p>Specify "sorts" in a space-separated list: sort1 sort2 ...</p> <p>An ordered list of sort criteria. Each list entry consists of a property name and a sort order--either an ascending or descending sort order. If you specify more than one sort criterion, the sorts are applied according to the order they appear in the list.</p> <p>The first item in the list is the primary sort key.</p> <p>Each item in the list is a hash reference.</p> <p>See the code example below for instructions on forming the list and passing it to the Commander Perl API.</p> <p>The sort order affects which objects are deleted if a maxIds value limits the number of objects returned by the filter.</p>

Positional arguments

objectType

Response

Returns a list of object references.

ec-perl

syntax: \$cmdr->deleteObjects(<objectType>, {<optionals>});

Example

This code example illustrates using a Boolean filter for the deleteObjects command to find jobs matching either of two patterns for the job name.

```
my @filterList;
push (@filterList, {"propertyName" => "jobName",
                    "operator" => "like",
                    "operand1" => "%-branch-%"});
push (@filterList, {"propertyName" => "jobName",
                    "operator" => "like",
```

```
        "operand1" => "branch-%"});  
my $result = $cmdr->deleteObjects('job',  
    {filter => [  
        { operator => 'or',  
          filter => \@filterList,  
        }  
    ]}  
);  
print "result = " . $result-> findnodes_as_string("n"). "\n";
```

ectool

Not supported.

[Back to Top](#)

export

Exports part or all server data to an XML file. By default, all data in the system is exported, although the "path" option can be used to limit the output to a single tree of objects.

If a relative filename is specified, the file is created relative to the Commander server's data directory, which by default is located:

For Windows: C:\Documents and Settings\All Users\Application Data\Electric Cloud\
ElectricCommander

For Linux: /opt/electriccloud/electriccommander

You must specify a `fileName`.

Note: A full export/import preserves job IDs, but a partial import preserves names only, not IDs.

Arguments	Descriptions
compress	<i><Boolean flag - 0 1 true false></i> Use this argument to compress XML output. If set to 1, the file will be compressed using the "gzip" format and a ".gz" file extension will be added to the filename. The default behavior is to compress the output. Note: This is true for full exports only, not a partial export.
excludeJobs	<i><Boolean flag - 0 1 true false></i> If set to 1, no job information will be exported. This argument can be used to reduce the size of the export file.

Arguments	Descriptions
<code>fileName</code>	<code><remoteFileName></code> The specified directory for the file must already exist in the system. If the path is local, it will be created on the server. If it is a network path, it must be accessible by the server and the server user.
<code>path</code>	<code><property path></code> Specifies the path for an object to be exported. Any single object can be exported if it is specified using property path syntax. The object and its sub-objects are exported.
<code>relocatable</code>	<p><code><Boolean flag - 0 1 true false></code> If the <code>--relocatable</code> flag is set to "true", a partial export (for example, with <code>--path</code>) will not include object IDs, ACLs, system property sheets, create/modify times, owners, email notifiers or <code>lastModifiedBy</code> information, and the export file result will be much smaller than a normal export. When this file is imported, the result should show one or more objects owned by the importing user as if they were newly created.</p> <p>Note: The <code>relocatable</code> argument ONLY works with a partial export. This argument is silently ignored during a full export.</p>
<code>safeMode</code>	<p>The <code>safeMode</code> argument determines whether the server will be quiesced before a full export begins and if yes, whether or not the server will shutdown and restarted after the export completes. Values are:</p> <ul style="list-style-type: none"> • none (default) - Do not quiesce the server during export. • shutdown - Quiesce the server and shutdown when complete. • restart - Quiesce the server and restart when complete. <p>Note: The <code>safeMode</code> argument has no effect on partial exports.</p>
<code>withAcls</code>	<p>Modifies <code>relocatable</code>.</p> <p><code><Boolean flag - 0 1 true false></code> If the <code>withAcls</code> flag is set to "true", a relocatable partial export will include ACLs.</p>
<code>withNotifiers</code>	<p>Modifies <code>relocatable</code>.</p> <p><code><Boolean flag - 0 1 true false></code> If the <code>withNotifiers</code> flag is set to "true", a relocatable partial export will include email notifiers.</p>

Positional arguments

`fileName`

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->export(<fileName>, {<optionals>});

Examples

```
$cmdr->export("c:\CommanderBackup\Mar 15 2007.xml");
```

```
$cmdr->export("c:\CommanderBackup\Test Proj.xml",  
             {path => "/projects[Test Proj]",  
              relocatable => "true",  
              withNotifiers => "true"});
```

ectool

syntax: ectool export <fileName> ...

Examples

```
ectool export "c:\CommanderBackup\Mar 15 2007.xml"
```

```
ectool export "c:\CommanderBackup\Test Proj.xml" --path "/projects[Test Proj]"  
--relocatable true --withNotifiers true
```

[Back to Top](#)

findObjects

This command returns a sorted list of Commander objects based on an object type and a set of filter criteria. This API can be used to find many, but not all, types of Commander objects and is used by the Commander web interface to implement the Commander "Search" feature.

Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.

You must specify an `objectType`.

Arguments	Descriptions
filter	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>Each element of the filter list is a hash reference containing one filter criterion. You can specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p> <p>Two types of filters:</p> <p>"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p>"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.</p> <p>Property filter operators are:</p> <pre> between (2 operands) contains (1) equals (1) greaterOrEqual (1) greaterThan (1) in (1) lessOrEqual (1) lessThan (1) like (1) notEqual (1) notLike (1) isNotNull (0) isNull (0) </pre> <p>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.</p> <p>Boolean operators are:</p>

Arguments	Descriptions																										
	<p>not (1 operand)</p> <p>and (2 or more operands)</p> <p>or (2 or more operands)</p>																										
maxIds	<p><id count></p> <p>The maximum number of object IDs that will be returned. If omitted, default behavior returns IDs for the first 1000 objects matching the query. If "0" is specified, the ID of every matching object is returned.</p>																										
numObjects	<p><full object count></p> <p>Specifies the number of full objects (not just the IDs) returned from the <code>findObjects</code> request. This option allows selecting a limited number of full objects to be returned in the initial request. The returned "full objects" correspond to the objects from the beginning of the list of object IDs. If <code>numObjects</code> is not specified, all full objects in the list of object IDs are returned. Any and all objects can be retrieved using the <code>getObjects</code> command.</p>																										
objectType	<p>This argument specifies the type of object to find.</p> <p>Values include:</p> <table> <tr> <td>artifact</td><td>project</td></tr> <tr> <td>artifactVersion</td><td>property</td></tr> <tr> <td>credential</td><td>repository</td></tr> <tr> <td>directoryProvider</td><td>resource</td></tr> <tr> <td>emailconfig</td><td>resourcePool</td></tr> <tr> <td>emailNotifier</td><td>schedule</td></tr> <tr> <td>formalParameter</td><td>state</td></tr> <tr> <td>job</td><td>stateDefinition</td></tr> <tr> <td>jobStep</td><td>transition</td></tr> <tr> <td>logEntry</td><td>transitionDefinition</td></tr> <tr> <td>plugin</td><td>workflow</td></tr> <tr> <td>procedure</td><td>workflowDefinition</td></tr> <tr> <td>procedureStep</td><td>workspace</td></tr> </table>	artifact	project	artifactVersion	property	credential	repository	directoryProvider	resource	emailconfig	resourcePool	emailNotifier	schedule	formalParameter	state	job	stateDefinition	jobStep	transition	logEntry	transitionDefinition	plugin	workflow	procedure	workflowDefinition	procedureStep	workspace
artifact	project																										
artifactVersion	property																										
credential	repository																										
directoryProvider	resource																										
emailconfig	resourcePool																										
emailNotifier	schedule																										
formalParameter	state																										
job	stateDefinition																										
jobStep	transition																										
logEntry	transitionDefinition																										
plugin	workflow																										
procedure	workflowDefinition																										
procedureStep	workspace																										
select	<p>This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</p>																										

Arguments	Descriptions
sort	This is an ordered list of sort criteria. Each list entry consists of a property name and a sort order—either an ascending or descending sort order. If you specify more than one sort criterion, the sorts are applied according to the order they appear in the list. The first item in the list is the primary sort key. Each item in the list is a hash reference. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.

Positional arguments

objectType

Response

This command returns a list of object references. These references can be used in a subsequent call to the [getObjects](#) command. Optionally, the command can return full objects from the result list also.

ec-perl

syntax: \$cmdr->findObjects(<objectType>, {<optionals>});

Example 1

This example illustrates using a Boolean filter for the `findObjects` command to find jobs matching either of two patterns for the job name.

```
my @filterList;
push (@filterList, {"propertyName" => "jobName",
                    "operator" => "like",
                    "operand1" => "%-branch-%"});
push (@filterList, {"propertyName" => "jobName",
                    "operator" => "like",
                    "operand1" => "branch-%"});
my $result = $cmdr->findObjects('job',
    {filter => [
        { operator => 'or',
          filter => \@filterList,
        }
    ]}
);
print "result = " . $result->findnodes_as_string("/") . "\n";
```

Example 2

This example uses both `findObjects` and `getObjects` to manage large result sets, and also uses "select" to return the values of two properties in the returned objects.

```
# Search for the first 10 matching objects and retrieve the first 2
my $xPath = $cmdr->findObjects("schedule",
    {maxIds => "10",
     numObjects => "2",
```

```
filter => [{propertyName => "createTime",
            operator => "greaterOrEqual",
            operand1 => "2007-01-20T00:00:00.000Z"},
           {propertyName => "lastModifiedBy",
            operator => "like",
            operand1 => "adm%"}],
sort => [{propertyName => "projectName",
          order => "ascending"},
         {propertyName => "createTime",
          order => "descending"}],
select => [{propertyName => 'prop1'},
           {propertyName => 'prop2'}]
});
print "Return data from Commander:\n" . $XPath-> findnodes_as_string("/"). "\n";
# Build a list of all the object id's
my @allObjectsList;
my $nodeset = $XPath->find('//response/objectId');
foreach my $node ($nodeset->get_nodelist)
{
    my $objectId = $node-> string_value();
    push (@allObjectsList, $objectId);
}
# Retrieve the second 2 objects
my @objectList = @allObjectsList[2..3];
$XPath = $cmdr->getObjects(
    {objectId => \@objectList});
print "Return data from Commander:\n" . $XPath->findnodes_as_string("/"). "\n";
```

Example 3

This code example illustrates composing filters combining 'or' and 'and' for finding artifacts matching either of two patterns for the artifact name, and a modify time before a specified date.

```
# Create the filter list for filtering on artifact name.
my @artifactNameFilters;
push (@artifactNameFilters,
    {"propertyName" => "artifactName",
     "operator" => "equals",
     "operand1" => "groupId:installer-windows"},
    {"propertyName" => "artifactName",
     "operator" => "equals",
     "operand1" => "groupId:installer-linux"}
);
# Perform the findObjects query
my $result = $cmdr->findObjects('artifactVersion',
    {filter =>
        {operator => "and", # 'and' the different filters below
          filter => [
              #filter 1
              {
                  propertyName => "modifyTime",
                  operator => "lessOrEqual", # Give me all dates before
                  operand1 => "2011-11-01T00:00:00.000Z" # 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 => "2011-11-01T00:00:00.000Z" # Arbitrary date
        },
        #filter 2
        {
            operator => 'or', # apply 'or' for the filters in the list
            filter => \@artifactNameFilters
        }
    ]
});

```

Example 4

This example illustrates looking for a project whose name contains 'foo' and whose description equals 'bar'.

```

$commander->findObjects('project', {
    filter => {operator => 'and',
        filter => [{propertyName => 'projectName',
            operator => 'contains',
            operand1 => 'foo'},
            {propertyName => 'description',
            operator => 'equals',
            operand1 => 'bar'}]}});

```

Example 5

This example illustrates looking for a procedure whose project name is 'foo' and whose procedure name is either 'bar' or not 'bar'. (The top level filters are implicitly combined with 'and'.)

```

$commander->findObjects('procedure', {
    filter => [{propertyName => 'projectName',
        operator => 'equals',
        operand1 => 'foo'},
        {operator => 'or',
            filter => [{propertyName => 'procedureName',
                operator => 'equals',
                operand1 => 'bar'},
                {operator => 'not',

```

```
filter      => {propertyName => 'procedureName',  
operator    => 'equals',  
operand1    => 'bat'}}}}}}};
```

Example 6

This example illustrates looking for a project with certain property values.

```
$commander->findObjects("project", {  
  filter => {operator => 'or',  
    filter => [{propertyName => 'prop1',  
      operator    => 'equals',  
      operand1    => 'value1'},  
    {propertyName => 'prop2',  
      operator    => 'equals',  
      operand1    => 'value2'},  
    {propertyName => 'prop3',  
      operator    => 'isNull'}]}
```

ectool

Not supported.

[Back to Top](#)

getObjects

The `getObjects` command retrieves a list of full objects based on object IDs returned by `findJobSteps` or `findObjects`. All requested objects must be of the same `objectType`. See [findObjects](#) for a list of object types.

You must specify `objectId`.

Arguments	Descriptions
<code>objectId</code>	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 Commander Perl API.
<code>select</code>	This is an unordered list of projection definitions. Each list entry consists of a property name identifying a top-level custom property to return in addition to the rest of the object elements. See the code example below for instructions on forming the list and passing it to the Commander Perl API.

Positional arguments

`objectId`

Response

A list of full objects for the requested type.

ec-perl

syntax: \$cmdr->getObjects({<optionals>});

Example 1

Code example for findObjects and getObjects:

```
# This example runs within a Commander step, so a "login" is not needed.
use strict;
use ElectricCommander;
my $cmdr = ElectricCommander->new();

# Search for the first 10 matching objects and retrieve the first 2
my $xPath = $cmdr->findObjects("schedule",
    {maxIds      => "10",
     numObjects => "2",
     filter => [{propertyName => "createTime",
                  operator    => "greaterOrEqual",
                  operand1    => "2010-01-20T00:00:00.000Z"},
                {propertyName => "lastModifiedBy",
                  operator    => "like",
                  operand1    => "adm%"}],
     sort => [{propertyName => "projectName",
                  order      => "ascending"},
              {propertyName => "createTime",
                  order      => "descending"}],
     select => [{propertyName => 'prop1'},
                {propertyName => 'prop2'}]
    });
print "Return data from Commander:\n" . $xPath-> findnodes_as_string("/"). "\n";
# Build a list of all the object id's
my @allObjectsList;
my $nodeset = $xPath->find('//response/objectId');
foreach my $node ($nodeset->get_nodelist)
{
    my $objectId = $node-> string_value();
    push (@allObjectsList, $objectId);
}

# Retrieve the second 2 objects
my @objectList = @allObjectsList[2..3];
$xPath = $cmdr->getObjects(
    {objectId => \@objectList});
print "Return data from Commander:\n" . $xPath-> findnodes_as_string("/"). "\n";
```

Example 2

Code example using a Boolean filter:

```
my $xpath = $N->findObjects('project', {
    filter => {operator => 'and',
              filter => [{propertyName => 'projectName',
                          operator    => 'contains',
                          operand1    => $projectBase},
```

```
{propertyName => 'description',  
  operator => 'equals',  
  operand1 => 'foo'}}}});
```

ectool

Not supported.

[Back to Top](#)

import

Imports data from an XML export file.

You must specify either `file` or `fileName`.

Note: A full export/import preserves job IDs, but a partial import preserves names only, not IDs.

Use the `preserveId` option for a partial import if you need to retain the same (existing) job or workflow ID number.

Arguments	Descriptions
<code>batchSize</code>	<p><i><batch size></i> The number of objects imported before committing a transaction in the database. This argument limits the object batch size during import. Default value is 50 objects. If your objects are unusually large, you can throttle this number down to 1, depending on your available memory.</p> <p>Note: The <code>batchSize</code> argument is applicable to full import operations only.</p>
<code>disableSchedules</code>	<p><i><Boolean flag - 0 1 true false></i> If set to 1, imported schedules will be disabled. This argument can modify imported schedules after import and before they are used to start a job.</p>
<code>file</code>	<p><i><localFileName></i> This is the path to a file on the client to import. The file is uploaded from the client to the server. The specified <i><file></i> value is sent as an attachment to the <code>import</code> API request. The server detects the presence of the attachment and reads the attached file instead of looking for a file on the server. The maximum file size specified by <code>file</code> is determined by the maximum upload-size server setting.</p> <p>By default the limit is 50MB, so this option should be used only for individually exported objects, not a full system export.</p>

Arguments	Descriptions
fileName	<i><remoteFileName></i> This is the name of a file on the server to import. The file path name must be accessible to the server process on the server host.
force	<i><Boolean flag - 0 1 true false></i> This argument can be used to replace a single object if it already exists at the specified property path.
path	<i><property path></i> Use this argument to import a single object to a new location. For example, if a procedure was exported from "project A", this argument allows you to import it into "project B", but only if the export used the <code>path</code> option also.
preserveId	If performing a <i>partial</i> import, using this option preserves the original job ID or workflow ID.

Positional arguments

fileName

Response

None or a status OK message.

ec-perl

syntax examples: `$cmdr->import(<fileName>, {...});`
`$cmdr->import({file => <localFileName>, ...});`

Examples

```
$cmdr->import("/opt/TestProg.xml");

$cmdr->import({file => "c:\r.xml", path => "/projects[Test]"});
```

ectool

syntax examples: `ectool import <remoteFileName> ...`
`ectool import --file <localFileName>`

Examples

```
ectool import /mnt/backups/fullBackup.xml

ectool --file "c:\project.xml" --path "/projects[Test]"
```

[Back to Top](#)

API Response and Element Glossary

The first part of this help topic lists returned response container elements in alphabetical order. The Contents for each container element lists all or most of the possible returned response elements—both simple and subcontainer elements. Depending on your request, you may not see all elements in your response. If the value of an element is "empty," typically that element is omitted from the response.

Note: Elements annotated with an * (asterik) may appear multiple times in a response.

The second part of this help topic is an element glossary for all single or "leaf" elements and subcontainer elements. Click [here](#) to go to the glossary or notice that each response element is a link—each response element is linked directly to its glossary entry.

access

Contains the set of effective permissions for a user or a group.

Contents:

[changePermissionsPrivilege](#)
[executePrivilege](#)
[modifyPrivilege](#)
[readPrivilege](#)

aclEntry

Contains an ACE (access control list entry) on an object for a given principal.

Contents:

[aclEntryId](#)
[changePermissionsPrivilege](#)
[executePrivilege](#)
[modifyPrivilege](#)
[readPrivilege](#)
[principalName](#)
[principalType](#)

actualParameter

An `actualParameter` object provides the value for a parameter, which is passed to a procedure when it is invoked.

Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are

different

from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters

provide values to use at run-time.

Contents:

`actualParameterId`
`actualParameterName`
`createTime`
`modifyTime`
`value`

artifact

Contains elements to define the artifact. An artifact is specified by `groupId` and `artifactKey`.

The name of an artifact is in this form "`groupId:artifactKey`". An artifact contains a collection of `artifactVersions`.

Contents:

`artifactId`
`artifactKey`
`artifactName`
`artifactVersionNameTemplate`
`createTime`
`description`
`groupId`
`lastModifiedBy`
`modifyTime`
`owner`
`propertySheetId`

artifactVersion

A "concrete" version of an artifact that contains a collection of files stored in the artifact repository.

Contents:

<code>artifactKey</code>	<code>majorMinorPatch</code>
<code>artifactName</code>	<code>modifyTime</code>
<code>artifactVersionId</code>	<code>owner</code>

artifactVersionName	propertySheetId
artifactVersionState	publisherJobId
buildNumber	publisherJobName
createTime	publisherJobStepId
dependentArtifacts	qualifier
description	repositoryName
groupId	retrievers
lastModifiedBy	version

credential

Contains a stored credential. The password is returned for the `getFullCredential` API only.

Contents:

- credentialId
- credentialName
- createTime
- description
- lastModifiedBy
- modifyTime
- owner
- password
- projectName
- propertySheetId
- userName

databaseConfiguration

Contain configuration information about communicating with the database used to store server data.

Contents:

- batchRequests
- batchSize
- completeUserName
- customDatabaseDialect
- customDatabaseDriver
- customDatabaseUrl

databaseDialect
databaseDriver
databaseName
databaseType
databaseUrl
hostName
port
statementCacheSize
userName

directoryProvider

Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).

Contents:

commonGroupNameAttribute	modifyTime
createTime	name
description	owner
directoryProviderId	position
domainName	propertySheetId
emailAttribute	providerIndex
enableGroups	providerName
fullUserNameAttribute	providerType
groupBase	realm
groupMemberAttributes	url
groupMemberFilter	useSSL
groupNameAttribute	userBase
groupSearchFilter	userNameAttribute
lastModifiedBy	userSearchFilter
managerDn	userSearchSubtree

testDirectoryProvider

Contains the results of testing a directory provider configuration as a list of test result blocks.

Each block contains a result with details about any failures. The `findGroupsTest` block also includes a list of groups for the test user.

The `findUserTest` block includes information about the user or users that matched the test user name.

Contents:

```
findGroupsTest
  testResult
  details
  groupList
  group*
findUserTest
  testResult
  details
  userList
    userInfo*
    email
    fullUserName
    mutable
    providerName
userAuthenticationTest
  testResult
  details
```

emailConfig

Contains information about the configuration used to communicate with an email server.

Contents:

```
configName
createTime
description
emailConfigId
emailConfigName
lastModifiedBy
mailFrom
mailHost
mailPort
mailProtocol
```

mailUser
modifyTime
owner
propertySheetId

emailNotifier

Contains information about an email notifier.

Contents:

condition
configName
container
createTime
description
destinations
emailNotifierId
eventType
formattingTemplate
lastModifiedBy
modifyTime
notifierName
owner
propertySheetId

formalParameter

Contains information about a formal parameter.

Contents:

container
createTime
defaultValue
description
expansionDeferred
formalParameterId
formalParameterName
lastModifiedBy

`modifyTime`
`owner`
`required`
`type`

gateway

Contains information about a gateway.

Contents:

`createTime`
`description`
`gatewayDisabled`
`gatewayId`
`gatewayName`
`hostName1`
`hostName2`
`lastModifiedBy`
`modifyTime`
`owner`
`port1`
`port2`
`propertySheetId`
`resourceName1`
`resourceName2`

group

Contains information about a defined group of users.

Contents:

`createTime`
`groupId`
`groupName`
`lastModifiedBy`
`modifyTime`
`mutable`
`owner`

propertySheet
 propertySheetId
 providerName
 users

job

Contains information about a running or completed job. Different API calls will result in different subsets of possible properties on the job. Refer to the specific API for details.

Contents:

abortedBy	licenseWaitTime
abortStatus	liveProcedure
actualParameters*	liveSchedule
callingState	modifyTime
combinedStatus	outcome
createTime	owner
credentialName	priority
deleted	procedureName
directoryName	projectName
elapsedTime	propertySheet
errorCode	propertySheetId
errorMessage	resourceWaitTime
external	runAsUser
finish	scheduleName
jobId	start
jobName	status
jobStep*	steps
lastModifiedBy	totalWaitTime
launchedByUser	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	outcome
abortStatus	owner
actualParameters	parallel
alwaysRun	postExitCode
assignedResourceName	postLogFileName
broadcast	postProcessor
calledProcedure	precondition
jobStep*	procedureName
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 Commander license.

Contents:

- `createTime`
- `customerName`
- `evaluation`
- `expirationDate`
- `featureName`
- `gracePeriod`
- `lastModifiedBy`
- `licenseId`
- `modifyTime`
- `owner`
- `productName`
- `property*`
- `propertySheet*`
- `signature`

licenseUsage

Contains information about Commander license usage.

Note: Your response will be different depending on how you are licensed for ElectricCommander currently.

Contents:

- `concurrentResources`
 - `inUseHosts`
 - `inUseProxiedHosts`
 - `maxHosts`
 - `maxProxiedHosts`
- `concurrentUsers*`
 - `adminLicenseLastUse`
 - `adminLicenseUser`
 - `inUseLicenses`
 - `maxLicenses`
 - `license*`
 - `admin`

- `expiration`
 - `lastUse`
 - `user`
- `concurrentSteps`
- `maxConcurrentSteps`
- `runningSteps`

logEntry

Contains information about log events generated anywhere in the system.

Contents:

- `category`
- `container`
- `containerName`
- `deleted`
- `logEntryId`
- `message`
- `principal`
- `severity`
- `subject`
- `subjectName`
- `time`

object

Primarily, the object element is returned from a `getAccess` API request. If multiple objects are returned, they are presented in an order beginning with the API requested object to the top-level object in the ACL hierarchy. Your object-query response can contain one or more `aclEntry` containers.

Contents:

- `objectId`
- `objectName`
- `objectType`
- `aclEntry*`

plugin

Contains elements to define the plugin.

Contents:

`author`
`createTime`
`description`
`label`
`lastModifiedBy`
`modifyTime`
`owner`
`pluginId`
`pluginKey`
`pluginName`
`pluginVersion`
`project`
`projectName`
`promoted`
`propertySheetId`

procedure

Contains elements to define the procedure.

Contents:

`attachedCredentials`
`createTime`
`credentialName`
`description`
`jobNameTemplate`
`lastModifiedBy`
`modifyTime`
`owner`
`procedureId`
`procedureName`
`projectName`
`propertySheetId`
`resourceName`
`workspaceName`

project

Contains all elements to define a project.

Contents:

- `attachedCredentials`
- `createTime`
- `credentialName`
- `deleted`
- `description`
- `lastModifiedBy`
- `modifyTime`
- `owner`
- `pluginName`
- `projectId`
- `projectName`
- `propertySheetId`
- `resourceName`
- `workspaceName`

property

Contains property sheets and various elements, depending on your query.

Contents:

- `createTime`
- `description`
- `expandable`
- `lastModifiedBy`
- `modifyTime`
- `owner`
- `path`
- `propertyId`
- `propertyName`
- `propertySheet*`
- `propertySheetId`
- `value`

propertySheet

Contains one or more property elements.

Contents:

`createTime`

`lastModifiedBy`

`modifyTime`

`owner`

`property*`

`propertySheetId`

repository

Contains elements to define the artifact repository. The most useful elements in this object are "repositoryName" and "url". Clients publishing/retrieving artifact versions search repositories by name to obtain connection information.

Contents:

`createTime`

`description`

`lastModifiedBy`

`modifyTime`

`owner`

`propertySheetId`

`repositoryDisabled`

`repositoryId`

`repositoryIndex`

`repositoryName`

`url`

`zoneName`

resource

Contains elements to define a resource.

Contents:

`agentState`

`lastRunTime`

`alive`

`modifyTime`

code	owner
details	pools
message	port
pingToken	propertySheetId
protocolVersion	proxyCustomization
state	proxyHostName
time	proxyPort
version	proxyProtocol
artifactCacheDirectory	repositoryNames
createTime	resourceDisabled
description	resourceId
exclusiveJobId	resourceName
exclusiveJobName	shell
exclusiveJobStepId	stepCount
exclusiveJobStepName	stepLimit
gateways	trusted
hostName	useSSL
hostOS	workspaceName
hostPlatform	zoneName
lastModifiedBy	

resourcePool

Contains elements to define a resource pool.

Contents:

- autoDelete
- createTime
- description
- lastModifiedBy
- lastResourceUsed
- modifyTime
- orderingFilter
- owner
- propertySheetId

resourceNames
resourcePoolDisabled
resourcePoolId
resourcePoolName

resourceUsage

Contains information about resource usage. For any step running on a resource, there is a resource usage record containing the ID and name of the job, job step, and resource.

Contents:

jobId
jobName
jobStepId
jobStepName
licenceWaitTime
resourceId
resourceName
resourcePoolId
resourcePoolName
resourceUsageId
resourceWaitTime
waitReason
workspaceWaitTime

schedule

Contains all elements to define a schedule.

Contents:

actualParameters	monthDays
attachedCredentials	owner
beginDate	priority
createTime	procedureName
credentialName	projectName
description	propertySheetId

endDate	scheduleDisabled
interval	scheduleId
intervalUnits	scheduleName
lastModifiedBy	startTime
lastRunTime	stopTime
misfirePolicy	timeZone
modifyTime	weekDays

serverStatus

Contains elements to determine the status of the server.

Contents:

```
apiMonitor
  longestCall
    api
    callId
    description
    elapsedTime
    label
    remoteAddress
    start
    userName
  mostActiveCalls
  totalCallCount
  activeCalls
    call*
      api
      callId
      description
      elapsedTime
      label
      remoteAddress
      start
      userName
  recentCalls
```


- `call*`
 - `api`
 - `callId`
 - `description`
 - `elapsedTime`
 - `label`
 - `remoteAddress`
 - `start`
 - `userName`
- `lastMessage`
- `messages`
 - `message*`
- `serverState`
- `startTime`

serverVersion

Contains elements to specify the Commander server version.

Contents:

- `label`
- `protocolVersion`
- `schemaVersion`
- `version`

state

Contains elements for a state in a running or completed workflow.

Contents:

- `active`
- `createTime`
- `description`
- `errorMessage`
- `index`
- `lastModifiedBy`
- `modifyTime`
- `owner`

projectName
propertySheetId
stateId
stateName
subjob
subprocedure
subproject
substartingState
subworkflow
subworkflowDefinition
workflowName

stateDefinition

Contains elements for the state definition within a workflow definition.

Contents:

createTime
description
formalParameters
index
lastModifiedBy
modifyTime
owner
projectName
propertySheetId
startable
stateDefinitionId
stateDefinitionName
subprocedure
subproject
substartingState
subworkflowDefinition
workflowDefinitionName

step

Contains elements to specify or define a step.

Contents:

actualParameters	postLogFileName
alwaysRun	postProcessor
attachedCredentials	precondition
attachedParameters	procedureName
broadcast	projectName
command	propertySheetId
condition	releaseExclusive
createTime	releaseMode
credentialName*	resourceName
description	shell
errorHandling	stepId
exclusive	stepName
exclusiveMode	subprocedure
lastModifiedBy	subproject
logFileName	timeLimit
modifyTime	timeLimitUnits
owner	workingDirectory
parallel	workspaceName

transition

Contains elements about a transition in a running or completed workflow.

Contents:

actualParameters
condition
createTime
description
index
lastModifiedBy
modifyTime

owner
projectName
propertySheetId
stateName
targetState
transitionId
transitionName
trigger
workflowName

transitionDefinition

Contains elements about a transition definition within a workflow definition.

Contents:

actualParameters
condition
createTime
description
index
lastModifiedBy
modifyTime
owner
projectName
propertySheetId
stateDefinitionName
targetState
transitionDefinitionId
transitionDefinitionName
trigger
workflowDefinitionName

user

Contains information about the current user.

Contents:

createTime

email
fullUserName
groups
lastModifiedBy
modifyTime
mutable
owner
propertySheetId
providerName
userId
userName

workflow

Contains elements about a running or completed workflow.

Contents:

activeState
callingState
completed
createTime
deleted
elapsedTime
finish
lastModifiedBy
launchedByUser
liveWorkflowDefinition
modifyTime
owner
projectName
propertySheetId
start
startingState
workflowDefinitionName
workflowId
workflowName

workflowDefinition

Contains elements about a workflow definition.

Contents:

`createTime`
`description`
`lastModifiedBy`
`modifyTime`
`owner`
`projectName`
`propertySheetId`
`workflowDefinitionId`
`workflowDefinitionName`
`workflowNameTemplate`

workspace

Contains elements about a workspace.

Contents:

`agentDrivePath`
`agentUncPath`
`agentUnixPath`
`createTime`
`credentialName`
`description`
`lastModifiedBy`
`local`
`modifyTime`
`owner`
`propertySheet`
`propertySheetId`
`workspaceDisabled`
`workspaceId`
`workspaceName`
`zoneName`

zone

Contains elements about a zone.

Contents:

`createTime`

`description`

`lastModifiedBy`

`modifyTime`

`owner`

`propertySheetId`

`resources`

`zoneId`

`zoneName`

Element Glossary

The following table lists all simple returned elements, including the element type and its description.

Returned element	Type	Description/Value
<code>abortStatus</code>	enum	Possible values are: <code>abort</code> <code>force_abort</code>
<code>abortedBy</code>	string	The name of the user who aborted the job.
<code>aclEntryId</code>	number	The unique Commander-generated ID for this <code>aclEntry</code> object.
<code>active</code>	boolean	<i><Boolean flag - 0 1 true false></i> —If set to "true", the state of the workflow is active.
<code>activeCalls</code>	subcontainer	A container element within the <code>serverStatus</code> element. <code>activeCall</code> describes an API currently running on the server.
<code>activeState</code>	string	The name of the <code>activeState</code> on the workflow object.

Returned element	Type	Description/Value
<code>actualParameters</code>	<code>propertySheet</code>	<p>An <code>actualParameter</code> object provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job.</p> <p>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.</p> <p>For the workflow feature, these are the parameters that were passed when the workflow was started.</p>
<code>actualParameterId</code>	<code>number</code>	The unique Commander-generated ID for this actual parameter object.
<code>actualParameterName</code>	<code>string</code>	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.
<code>admin</code>	<code>boolean</code>	<i><Boolean flag - 0 1 true false></i> —If set to "true", the this is an "admin" license.
<code>adminLicenseLastUse</code>	<code>date</code>	The time at which the admin license was last used.
<code>adminLicenseUser</code>	<code>string</code>	The name of the user who is currently licensed as the "admin" user.
<code>agentDrivePath</code>	<code>string</code>	Drive-letter-based path used by Windows agents to access the workspace in steps.
<code>agentUncPath</code>	<code>string</code>	UNC path used by Windows Commander Web servers to access the workspace. The agent uses <code>agentUncPath</code> and <code>agentDrivePath</code> to compute the drive mapping needed for making <code>agentDrivePath</code> valid in the step.

Returned element	Type	Description/Value
agentUnixPath	string	UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.
agentState	subcontainer	A subcontainer element returned from certain <code>resource queries</code> . <code>agentState</code> returns specific information about an agent, including the state of the agent. Possible values are: <code>unknown alive down</code>
alive	boolean	Refers to the agent state or status.
alwaysRun	boolean	<Boolean flag - 0 1 true false> - If set to 1, indicates this step will run even if the job is aborted before the step completes. Defaults to "false".
api	string	An element returned on <code>longestCall</code> , <code>activeCall</code> , and <code>recentCall</code> subcontainers of the <code>serverStatus</code> element. <code>api</code> returns the API call (command) that is running or ran on the server.
apiMonitor		A server object that tracks API active and recent calls, as well as the total number of calls since server startup.
artifactCacheDirectory	string	The directory on the agent host where retrieved artifacts are stored.
artifactId	number	The unique Commander-generated ID for this artifact object.
artifactKey	string	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName	string	The name of the artifact.
artifactsDirectory	string	The directory in the workspace where you can put files to view, using a report link.

Returned element	Type	Description/Value
artifactVersionId	string	The unique Commander-generated ID for this artifact version object.
artifactVersionName	name	The name of the artifact version. An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as " <code>groupId:artifactKey:version</code> " and the object is searched either way you specify its name—the Commander server interprets either name form correctly.
artifactVersionNameTemplate	string	A template for the names of artifact versions published to this artifact. Over-rides the global <code>artifactVersionNameTemplate</code> . The global setting can be manipulated in the Server Settings page (Administration > Server, select the Settings link).
artifactVersionState	enum	Possible values are: <code>available publishing unavailable</code>
assignedResourceName	string	The name of the resource assigned to the step by the step scheduler.
attachedCredentials	list	The names of the credentials attached to the specified object.
attachedParameters	string	These are credential parameters that were attached to a step.
author	string	The author of the plugin.
autoDelete	boolean	<i><Boolean flag - 0 1 true false></i> - If "true", the resource pool is deleted when the last resource is removed or deleted.
batchRequests	string	A setting in the database configuration that determines whether or not to batch SQL queries when making a request to the database.

Returned element	Type	Description/Value
batchSize	string	The number of objects imported before being committed to the database.
beginDate	string	<yyyy-mm-dd> The date the schedule is set to begin.
broadcast	boolean	<Boolean flag - 0 1 true false> - Used for command steps, this flag is used to run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the <code>resourceName</code> argument. Defaults to "false".
buildNumber	string	User-defined build number component of the version attribute for the artifact version.
call	subcontainer	A subcontainer returned on <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.
callId	number	A unique Commander-generated identifier for this particular call.
callingState	string	The full property path to the "calling state", which can appear on <code>subjobs</code> and <code>subworkflows</code> of a workflow.
calledProcedure	list	A subcontainer element within the <code>jobStep</code> element. The <code>calledProcedure</code> element can contain multiple <code>jobStep</code> elements.
category		(currently not used)
changePermissionsPrivilege	enum	Possible values are: <code>allow deny inherit</code>
code	enum	Script to execute the functions for a step—passed to the step's shell for execution.
combinedStatus	enum	More inclusive step status output - this value may combine up to three sub-elements: <code>status message properties</code>

Returned element	Type	Description/Value
<code>command</code>	string	The command to run steps - for command steps.
<code>commonGroupNameAttribute</code>	string	The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider.
<code>completed</code>	boolean	<i><Boolean flag - 0 1 true false></i> - If "true", the workflow is completed and no additional transactions will be evaluated.
<code>completeUserName</code>	string	A SQL server-specific tag that includes the user's name and the user's domain name.
<code>concurrentResources</code>	object	A subcontainer element that includes information about "in use" and "maximum licensed" hosts and proxied hosts for the <code>licenseUsage</code> API command.
<code>concurrentSteps</code>	number	The total number of steps running at the same time in the Commander system. This means all steps from all procedures, regardless of how many or how few projects you have created.)
<code>concurrentUsers</code>	object	A subcontainer element that includes information about the admin license, "in use" licenses, and the maximum number of licenses for the <code>licenseUsage</code> API command.

Returned element	Type	Description/Value
<code>condition</code>	string	<p>For steps: If empty or non-zero, the step will run. If set to "0", the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.</p> <p>For email notifiers: Mail sent if the condition evaluates to "true". The <code>condition</code> is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no <code>condition</code> is specified, the notification is ALWAYS sent.</p>
<code>configName</code>	string	The name of the configuration.
<code>container</code>	string	<p>An object ID for a "container" that contains formal parameters.</p> <p>In another context, this is typically the type and name of the workflow or job with a corresponding ID.</p>
<code>containerName</code>	string	The name of the container.
<code>createTime</code>	date	The time when this object was created.
<code>credentialId</code>	number	The unique Commander-generated ID for this credential object.
<code>credentialName</code>	string	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.</p> <p>absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>) - the credential can be from any specified project, regardless of the target object's project.</p>

Returned element	Type	Description/Value
<code>customDatabaseDialect</code>	string	Class name for the Hibernate dialect. The server chooses an appropriate dialect based on <code>databaseType</code> or this can be part of the custom specification.
<code>customDatabaseDriver</code>	string	Class name of the JDBC driver. The server will choose an appropriate driver based on <code>databaseType</code> or this can be part of the custom specification.
<code>customDatabaseUrl</code>	string	The JDBC URL to use. The server will compose an appropriate URL or this can be part of the custom specification.
<code>customerName</code>	string	The name of a company and/or group name with a company that is using ElectricCommander.
<code>databaseDialect</code>	string	Class name for the Hibernate dialect (the server chooses an appropriate dialect based on <code>databaseType</code>).
<code>databaseDriver</code>	string	Class name of the JDBC driver (the server will choose an appropriate driver based on <code>databaseType</code>).
<code>databaseName</code>	string	The name of the database the Commander server is using.
<code>databaseType</code>	enum	Possible values are: <code>builtin mysql oracle postgresql sqls server</code>
<code>databaseUrl</code>	string	The JDBC URL to use (the server will compose an appropriate URL).
<code>defaultValue</code>	string	This value is used for the formal parameter if a value is not supplied by the caller.
<code>delayUntil</code>	date	For a step that was rescheduled due to a resource or workspace problem, this is the next time when the step will be eligible to run.

Returned element	Type	Description/Value
deleted	byte	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
dependentArtifacts	string	A space-separated list of artifacts.
description	string	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
destinations	string	A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.
details	string	A string containing details about agent status.
directoryName	string	The name of the job's directory within each workspace for a job.
directoryProviderId	number	The unique Commander-generated ID for this directory provider object.
domainName	string	The name of the domain from which the Active Directory server(s) are automatically discovered.
elapsedTime	number	The number of milliseconds between the start and end times for the job or job step - or a workflow.
email	string	The user's email address.
emailAttribute	string	The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.

Returned element	Type	Description/Value
emailConfigId	number	The unique Commander-generated ID for this email configuration object.
emailConfigName	string	The name of the email configuration.
emailNotifierId	number	The unique Commander-generated ID for this email notifier object.
enableGroups	boolean	Determines whether or not to enable external groups for the directory provider.
endDate	string	<yyyy-mm-dd> The date this schedule is set to end.
errorCode	enum	Displays the error code, identifying which error occurred.
errorHandling	enum	<p>Determines what happens to the procedure if the step fails:</p> <ul style="list-style-type: none"> • <code>failProcedure</code> - The current procedure continues, but the overall status is error (default). • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - Continues as if the step succeeded.
errorMessage	string	A description of the error.
evaluation	boolean	Determines whether or not this license is an evaluation copy only.

Returned element	Type	Description/Value
eventType	enum	Possible values are: <code>onCompletion onStart</code> "onStart" triggers an event when the job or job step begins. "onCompletion" triggers an event when the job finishes, no matter how it finishes. Default is "onCompletion".
exclusive	boolean	<Boolean flag - 0 1 true false> - If set to 1, indicates this step should acquire and retain this resource exclusively. Defaults to "false".
exclusiveJobId	number	The ID number of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobName	string	The name of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobStepId	number	The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
exclusiveJobStepName	name	The name of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
exclusiveMode	enum	Possible values are: <code>none job step call</code> See exclusive
executePrivilege	enum	Possible values are: <code>allow deny inherit</code>
exitCode	number	The step's exit code.
expandable	boolean	<Boolean flag - 0 1 true false> Determines whether the property value will undergo property expansion when it is fetched. Default is "true".

Returned element	Type	Description/Value
expansionDeferred	boolean	<i><Boolean flag -0 1 true false></i> Default is "false," which means the formal parameter is expanded immediately.
expiration	date	The date when a user license expires.
expirationDate	date	The date when a license expires.
external	boolean	<i><Boolean flag -0 1 true false></i> If "true," this job is external. For more information about external jobs, see the API Commands - Job Management Help topic.
featureName	string	The name of the licensed feature. Possible features include: <code>Server</code>
findGroupsTest	subcontainer	For the <code>testDirectoryProvider</code> API, this element provides information on which groups the user is a member.
findUserTest	subcontainer	For the <code>testDirectoryProvider</code> API, this element contains specific information about the user.
finish	date	The time the job or workflow completed.
formalParameterId	number	The formal parameter's ID.
formalParameterName	string	The name of the procedure's parameter, containing a credential reference.
formalParameters	string	The parameters that must be supplied when entering the state (similar to formal parameters on a procedure).
formattingTemplate	string	Specifies a template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code> .
fullUserName	string	The user's full name - not his or her nickname.

Returned element	Type	Description/Value
<code>fullUserNameAttribute</code>	string	The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.
<code>gatewayDisabled</code>	boolean	<i><Boolean flag -0 1 true false></i> If "true", the gateway is disabled.
<code>gatewayId</code>	number	The Commander-generated ID number for this gateway.
<code>gatewayName</code>	string	The name of the gateway.
<code>gateways</code>	list	A space-separated list of gateway names.
<code>gracePeriod</code>	number	The number of days available after the Commander license expires.
<code>groupBase</code>	string	This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
<code>groupId</code>	number	The unique Commander-generated group ID. For Artifact Management: A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>groupList</code>	list	For the <code>testDirectoryProvider</code> API, this element contains zero or more groups returned after querying existing groups known to the directory provider.
<code>groupMemberAttributes</code>	string	A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.

Returned element	Type	Description/Value
groupMemberFilter	string	This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupName	string	The full name of a group. For Active Directory and LDAP, this is a full DN.
groupNameAttribute	string	The group record attribute that contains the name of the group.
groups	list	A space-separated list of group names.
groupSearchFilter	string	The LDAP query performed in the context of the groups directory to enumerate group records.
groupSettingsId	number	The unique Commander-generated ID for this group settings object.
hostName	string	The computer name or IP address for the machine containing the Commander server or agent.
hostName1	string	For gateways: The name Resource 2 uses to communicate with Resource 1. If "blank", the <i>Agent Host Name</i> attribute in Resource 1's definition is used at runtime.
hostName2	string	For gateways: The name Resource 1 uses to communicate with Resource 2. If "blank", the <i>Agent Host Name</i> attribute in Resource 2's definition is used at runtime.

Returned element	Type	Description/Value
hostOS	string	The full name of the host operating system, plus its version. However, if this host is a proxy, the value is "proxied".
hostPlatform	string	Examples for "platform" are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, the value is "proxied".
index	number	The numeric index of the transition that indicates its order in the list of transitions in a state definition.
interval	string	The repeat interval for starting new jobs.
intervalUnits	enum	Possible values are: hours minutes seconds continuous If set to continuous, Commander creates a new job as soon as the previous job completes.
inUseHosts	number	The number of hosts (agents) currently in use.
inUseLicenses	number	The number of user licenses currently in use.
inUseProxiedHosts	number	The number of proxy target hosts currently in use.
jobId	number	The unique Commander-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
jobName	string	The name of the job.
jobNameTemplate	string	Template used to determine the default name of jobs launched from a procedure.
jobStepId	number	The unique identifier for a job step, assigned automatically when the job step is created.
jobStepName	string	The name of the job step.

Returned element	Type	Description/Value
<code>label</code>	string	A name used by a plugin for display in a list, or this may represent context-specific info about an API call—not all API calls return a "label" tag.
<code>lastMessage</code>	string	Element returned by the <code>serverStatus</code> API showing the last message the server received.
<code>lastModifiedBy</code>	string	Shows who (generally a user name) last modified the object.
<code>lastResourceUsed</code>	string	The name of the most recently used resource from the pool.
<code>lastRunTime</code>	date	The last time a job was launched by a schedule. -or- In a <code>resource</code> response, this is the most recent time that a job step ran on the resource.
<code>lastUse</code>		Returned element in the <code>concurrentUsers</code> subcontainer (for the <code>licenseUsage</code> API), providing the last time a specific user accessed Commander.
<code>launchedByUser</code>	string	The name of the user or project principal that explicitly launched the job. This property is blank when the job is launched by a schedule.
<code>licenseId</code>	number	The unique Commander-generated ID for this license.
<code>licenseWaitTime</code>		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.

Returned element	Type	Description/Value
<code>liveProcedure</code>	string	Shows the current procedure name for the procedure step from which the job or job step was created – if the procedure step was renamed since the job or job step was launched, this is the procedure step's new name, and if the procedure step was deleted, this will be null.
<code>liveProcedureStep</code>	string	Shows the current procedure step name for the procedure step from which the job step was created – if the procedure step was renamed since the job was launched, this is the procedure step's new name, and if the procedure step was deleted, this will be null.
<code>liveSchedule</code>	string	Shows the current schedule name for the procedure step from which the job was created – if the schedule was renamed since the job was launched, this is the schedule's new name, and if the schedule was deleted, this will be null.
<code>liveWorkflowDefinition</code>	string	Shows the current workflow definition name for the workflow definition from which the workflow was created – if the workflow definition was renamed since the workflow was launched, this is the workflow definition's new name, and if the workflow definition was deleted, this will be null.
<code>local</code>	boolean	<i><Boolean flag -0 1 true false></i> If "true", this object is local.
<code>logEntryId</code>	number	The Commander-generated ID number for the log entry record.
<code>logFileName</code>	string	A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.
<code>longestCall</code>	string	Provides the API call that took the longest time.

Returned element	Type	Description/Value
<code>mailFrom</code>	string	The email address used as the email sender address for notifications.
<code>mailHost</code>	string	The name of the email server host.
<code>mailPort</code>	number	The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTP). Specify a value for this argument when a non-default port is used.
<code>mailProtocol</code>	string	This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
<code>mailUser</code>	string	This can be an individual or a generic name like "Commander" - name of the email user on whose behalf Commander sends email notifications.
<code>majorMinorPatch</code>	string	<code>major.minor.patch</code> component of the version attribute for the artifact.
<code>managerDn</code>	string	The name of a user who has read-only access to the LDAP or Active Directory server. Typically a DN (distinguished name). A simple name may be used when the Active Directory server's URL is being auto-discovered via DNS. Note: This user does not need to be an admin user with modify privileges.
<code>maxConcurrentSteps</code>	number	The maximum number of steps that can run at the same time per the provisions of your Commander license.
<code>maxHosts</code>	number	The maximum number of hosts licensed for resource use.
<code>maxLicenses</code>	number	The maximum number of licenses available for users.
<code>maxProxiedHosts</code>	number	The maximum number of available licenses for proxy hosts.

Returned element	Type	Description/Value
message	string	A user-readable diagnostic message associated with an error.
messages	list	Multiple error or diagnostic messages.
misfirePolicy	enum	<p>Possible values are: <code>ignore</code> <code>run once</code></p> <p>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.</p> <p>When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is <code>'ignore'</code>, otherwise it will run the job immediately. Defaults to <code>"ignore"</code>.</p>
modifyPrivilege	enum	Possible values are: <code>allow</code> <code>deny</code> <code>inherit</code>
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 <code>"true,"</code> the member list of this group is editable within Commander via the web UI or the <code>modifyGroup</code> API.
name	string	The name of the directory provider.
notifierName	string	The name of the email notifier.
objectId	number	<p>An object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</p> <p>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.</p>
objectName	string	The name of the object.

Returned element	Type	Description/Value
objectType	enum	The type of object being described, for example: project, procedure, step, and so on.
orderingFilter	string	A Javascript block invoked when scheduling resources for a pool. Note: A Javascript block is not required unless you need to override the default resource ordering behavior.
outcome	enum	Possible values for <code>outcome</code> : Note: The <code>outcome</code> is accurate only if the job status is "completed." <code>success</code> - The job finished successfully. <code>warning</code> - The job completed with no errors, but encountered some suspicious conditions. <code>error</code> - The job has finished execution with errors.
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 <code>pingToken</code> value is generated. The server uses the <code>pingToken</code> value to determine agent restarts by noticing the values before and after a restart.
pluginId	number	The unique Commander-generated ID for the plugin object.

Returned element	Type	Description/Value
<code>pluginKey</code>	string	The name of the plugin as displayed on the Commander Plugin Manager web page.
<code>pluginName</code>	string	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
<code>pluginVersion</code>	string	The version of the plugin being described.
<code>pools</code>	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.
<code>port</code>	number	If a port number is not specified, the default Commander port is used. For a proxy resource, this is the port number for the service running on the proxy target that will run commands on behalf of the ElectricCommander agent. For <code>ssh</code> , the default is 22.
<code>port1</code>	number	The port number used by <i>Gateway Resource 1</i> - default is to the port number used by the resource.
<code>port2</code>	number	The port number used by <i>Gateway Resource 2</i> - default is to the port number used by the resource.
<code>position</code>	number	Used to reorder a Commander object. For example, if reordering directory providers: the provider name is moved to a position just before this provider. "Blank" means move the provider to the end of the provider list.
<code>postExitCode</code>	number	The step's post processor exit code.
<code>postLogFileName</code>	string	The log file name produced by this step's post processor.

Returned element	Type	Description/Value
<code>postProcessor</code>	string	<p>This program looks at the step output to find errors and warnings. Commander includes a customizable program called “postp” for this purpose.</p> <p>The value for <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).</p>
<code>precondition</code>	string	<p>Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a “\0” or “false” is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p>
<code>principal</code>	string	The user or project principal from the session that was active when the event occurred.
<code>principalName</code>	string	This is either a user or a group name.
<code>principalType</code>	enum	Possible values are: <code>group user</code>
<code>priority</code>	enum	<p>Possible values are: <code>low normal high highest</code></p> <p>Priorities take effect when two or more job steps in different jobs are waiting for the same resource.</p> <p>When the resource is available, it will be used by the job step that belongs to the job with the highest priority.</p> <p>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.</p> <p>If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.</p>

Returned element	Type	Description/Value
procedureId	number	The unique Commander-generated procedure ID.
procedureName	string	The name of the procedure - may be a path to the procedure.
productName	string	The name of the product with the licensed feature. Possible products include: ElectricCommander
project	name	The name of the project associated with the plugin.
projectId	number	The unique Commander-generated project ID.
projectName	string	The name of the project - may be a path. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.
promoted	boolean	<i><Boolean flag - 0 1 true false></i> The new value of the promoted flag for the specified plugin. Default is "true", which means the plugin will be promoted. If you want to demote the plugin, use the value of "0" or false.
propertyId	number	The unique Commander-generated property ID.
propertyName	string	The name of the property. It may be a relative or absolute property path, including "my" paths such as "/myProject/prop1".
propertySheetId	number	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
protocolVersion	string	The server API protocol version. For example, the server accepts messages from ectool and ec-perl.

Returned element	Type	Description/Value
<code>providerIndex</code>	number	The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index "0" and one with index "1" means the providers will be searched in that numerical order.
<code>providerName</code>	string	The LDAP or Active Directory provider name.
<code>providerType</code>	enum	Possible values are: <code>ldap activedirectory</code>
<code>proxyCustomization</code>	string	Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.
<code>proxyHostName</code>	string	The name or IP address of the computer containing the ElectricCommander Agent used for a proxy resource.
<code>proxyPort</code>	number	The ElectricCommander agent port number for a proxy resource.
<code>proxyProtocol</code>	string	Protocol for communicating with the proxy target. Defaults to <code>ssh</code> . (This argument is not exposed in the ElectricCommander Web Interface at this time.)
<code>publisherJobId</code>	number	The Commander-generated ID for the job that published the artifact version.
<code>publisherJobName</code>	name	The name of the job that published the artifact version.
<code>publisherJobStepId</code>	number	The Commander-generated ID for the job step that published the artifact version.
<code>qualifier</code>	string	User-defined qualifier component of the version attribute for the artifact.
<code>readPrivilege</code>	enum	Possible values are: <code>allow deny inherit</code>

Returned element	Type	Description/Value
realm	string	The realm of the LDAP directory provider—used to create unique user names when there are multiple providers.
recentCall	subcontainer	A subcontainer element on the <code>serverStatus</code> API - a call no longer active (completed). The API monitor keeps track of the 10 most recent calls.
releaseExclusive	boolean	<i><Boolean flag - 0 1 true false></i> Declares whether or not this step will release its resource, which is currently held exclusively.
releaseMode	string	Possible values are: <code>none release releasetojob</code>
remoteAddress	string	Generally a combined IP address plus a port specification - used when the agent is talking to the server or to show where the request to the server originated.
repositoryDisabled	boolean	<i><Boolean flag - 0 1 true false></i> Determines whether the repository is disabled. Default is "false".
repositoryId	number	The Commander-generated ID for the artifact repository.
repositoryIndex	integer	The order of the repository within a list of repositories.
repositoryName	string	The name of the artifact repository.
repositoryNames	list	A list of one or more repository server names—each repository name listed on a "new line".
required	boolean	<i><Boolean flag - 0 1 true false></i> If set to 1, this value indicates whether a non-blank value must be supplied when calling the procedure.

Returned element	Type	Description/Value
resourceDisabled	boolean	<Boolean flag - 0 1 true false> If set to 1, Commander will not start new steps on this resource. Defaults to "false".
resourceId	number	The unique Commander-generated ID for this resource.
resourceName1	string	The name for the first of two resources required to create a gateway. "Spaces" are NOT allowed in a resource name.
resourceName2	string	The name for the second of two resources required to create a gateway. "Spaces" are NOT allowed in a resource name.
resourceName	string	The name of a resource.
resourceNames	string	A list of strings that refer to resources that belong to the pool. Names that do not refer to existing resources are ignored.
resourcePoolDisabled	boolean	<Boolean flag - 0 1 true false> If set to 1, Commander will not use resources in this pool. Defaults to "false".
resourcePoolId	number	The unique ID number for a resource pool.
resourcePoolName	name	The name of the resource pool.
resources	string	A space-separated list of resource names.
resourceUsageId	number	The unique ID number of the resource usage record.
resourceWaitTime		The amount of time a job step waited for a resource to become available. On a job, this is the sum of time all job steps waited for resource availability. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.

Returned element	Type	Description/Value
<code>retries</code>	number	The number of attempts to write to the step log in the workspace. In a running step, this is the number of retries attempted up to this point. The most common reason for step retries is the workspace for the step was unavailable.
<code>retrievers</code>	list	A collection of retrieve elements that can contain a <code>jobName</code> , <code>jobId</code> , and/or a <code>jobStepId</code> element(s).
<code>runAsUser</code>	string	The name of the user being impersonated in this job.
<code>runnable</code>	date	The time when the step became runnable.
<code>runningSteps</code>		The number of steps running at the same time.
<code>runtime</code>	number	The number of milliseconds the step command spent running on a resource.
<code>scheduleDisabled</code>	boolean	<i><Boolean flag - 0 1 true false></i> If set to 1, ElectricCommander does not start any new jobs from the schedule. Defaults to "false".
<code>scheduleId</code>	number	The unique Commander-generated ID for the schedule.
<code>scheduleName</code>	string	The name of the schedule - may be a path to the schedule.
<code>schemaVersion</code>	number	The Commander server's database schema version.
<code>serverState</code>	enum	Possible values are: <code>bootstrap</code> , <code>databaseConfiguration</code> , <code>databaseConnection</code> , <code>databaseSchema</code> , <code>running</code> , <code>failed</code> , <code>stopping</code> , <code>importFailed</code>
<code>severity</code>	enum	Possible values are: <code>INFO</code> <code>WARN</code> <code>ERROR</code>

Returned element	Type	Description/Value
shell	string	Where shell is the name of a program used to execute commands contained in the "command" field. Normally, this file is a command shell, but it could be any other command line program. The default is "cmd /q /c" for a Windows agent and "sh -e" for a UNIX agent. This is applicable to command steps only.
signature	string	The digital signature on this license.
start	date	The time this job or workflow began executing.
startable	boolean	"True" means this state definition can be the initial state of an instantiated workflow.
startingState	string	The initial state of the workflow.
startTime	string	Formatted hh:mm, using the 24-hour clock. Using this schedule, Commander starts creating jobs at this time on the specified days.
stateDefinitionId	number	The unique Commander-generated ID for this state definition object.
stateDefinitionName	string	The name of the state definition.
stateId	number	The unique Commander-generated ID for this state object.
statementCacheSize	string	The number of MS SQL statements cached in the database.
stateName	string	The name of the state.

Returned element	Type	Description/Value
<code>status</code>	enum	Possible values for <code>status</code> : <code>pending</code> - The job is not yet runnable—it is waiting for other steps to complete first. <code>runnable</code> - The job is ready to run, but it is waiting for a resource to become available. <code>running</code> - The job is assigned to a resource and is executing the step command. <code>completed</code> - The job finished executing.
<code>stepCount</code>	number	The number of executing steps on this resource.
<code>stepErrorCode</code>	enum	Agent error messages.
<code>stepId</code>	number	The unique Commander-generated ID for the step.
<code>stepLimit</code>	number	The number of steps that can run on the resource at one time. (Previously setting the limit to 1 enforces serial access to the resource.)
<code>stepName</code>	string	The name of the step - may be a path to the step.
<code>steps</code>		The list or number of steps in a job.
<code>stopTime</code>	string	Formatted <code>hh:mm</code> , using the 24-hour clock. ElectricCommander stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, ElectricCommander creates one job only on each specified day.
<code>subject</code>	string	Refers to the object the event concerns (similar to <code>container</code>).
<code>subjectName</code>	string	The name of the subject/object.
<code>subjob</code>	string	The name of the subjob.

Returned element	Type	Description/Value
subprocedure	string	The name of the nested procedure called when a step runs. If a subprocedure is specified, <code>command</code> or <code>commandFile</code> options are not necessary.
subproject	string	If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	string	Name of the starting state for the workflow launched when the state is entered.
subworkflow	string	The name of the subworkflow.
subworkflowDefinition	string	The name of the subworkflow definition.
targetState	string	The target state for the transition definition.
testResult	enum	Possible values are: <code>success skipped failure</code>
time	date	The time of day to invoke this schedule's procedure (24-hour clock, for example, 17:00). For a <code>logEntry</code> response, <code>time</code> indicates the time at which data was written to the log.
timeLimit	number	The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.
timeLimitUnits	enum	Possible values are: <code>hours minutes seconds</code>
timeout	number	Specifies the timeout for the <code>element</code> flag. The default value is 120 seconds.
timeZone	string	The time zone specified to use for this schedule (Java-compatible string).

Returned element	Type	Description/Value
totalCallCount	number	The total number of API calls to the server since startup.
totalWaitTime		On a job, this is the sum of total time all job steps waited for license, resource, and/or workspace availability.
transitionDefinitionId	number	The unique Commander-generated ID for this transition definition.
transitionDefinitionName	string	The name of the transition definition.
transitionId	number	The unique Commander-generated ID for this transition object.
transitionName	string	The name of the transition.
trigger	enum	Possible values are: onEnter onStart onCompletion>manual
trusted	boolean	<p><Boolean flag - 0 1 true false> If "true", the resource is <i>trusted</i>. A trusted agent is one that has been "certificate verified."</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • trusted - the Commander server verifies the agent's identity using SSL certificate verification. • untrusted - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.
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.

Returned element	Type	Description/Value
<code>url</code>	string	<p>For directory providers: The server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas. Note: Spaces in the <code>basedn</code> must be URL encoded (<code>%20</code>). For artifact repositories: The server URL is in the form <code>protocol://host:port/</code>. Typically, the repository server is configured to listen on port 8200 for <code>https</code> requests, so a typical URL looks like <code>https://host:8200/</code>.</p>
<code>userAuthenticationTest</code>	subcontainer	For the <code>testDirectoryProvider</code> API, this element authenticates the user.
<code>userBase</code>	string	The string prepended to the <code>basedn</code> to construct the directory DN that contains user records.
<code>userId</code>	number	The unique Commander-generated ID for the user.
<code>userInfo</code>		<code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include multiple <code>userInfo</code> tags, each of which describes a user (including full name, email address, and provider name).
<code>userList</code>	list	<code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include one or more <code>userInfo</code> tags.
<code>userName</code>	string	The full name of the user. For Active Directory and LDAP, the name may be <code>user@domain</code> .

Returned element	Type	Description/Value
<code>userNameAttribute</code>	string	The attribute in a user record that contains the user's account name.
<code>userSearchFilter</code>	string	The LDAP query performed in the context of the user directory to search for a user by account name. The string "{0}" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
<code>userSearchSubtree</code>	boolean	<i><Boolean flag - 0 1 true false></i> If true, the subtree below the user base was recursively searched.
<code>userSettingsId</code>	number	The unique Commander-generated ID for the user settings.
<code>useSSL</code>	boolean	<i><Boolean flag - 0 1 true false></i> This flag is used to specify using SSL to communicate with your Active Directory servers.
<code>value</code>	string	For a string property, this is the value of the property. For a sheet property, this argument is invalid.
<code>version</code>	string	For plugin versions, the value is represented in the form: <code>major.minor.</code> For artifact versions, the value is represented in the form: <code>major.minor.patch-qualifier-buildNumber</code>
<code>waitReason</code>	string	Possible values are: <code>license, resource, or workspace</code> Generally, these objects are unavailable, causing a longer wait time for availability.
<code>waitTime</code>	number	The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).

Returned element	Type	Description/Value
weekDays	string	Restricts the schedule to specified days of the week. Days of the week are separated by spaces. English names "Monday", "Tuesday", and so on.
workflowDefinitionId	number	The unique Commander-generated ID for this workflow definition.
workflowDefinitionName	string	The name of the workflow definition.
workflowId	number	The unique Commander-generated ID for this workflow object.
workflowName	string	The name of this workflow.
workflowNameTemplate	string	Template used to determine the default names for workflows launched from a workflow definition.
workingDirectory	string	The Commander agent sets this directory as the "current working directory," when running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace as the working directory. Note: If running a step on a proxy resource, this directory must exist on the proxy target.
workspaceDisabled	boolean	<Boolean flag - 0 1 true false> - If "true," the workspace is disabled.
workspaceId	number	The unique Commander-generated ID for the workspace.
workspaceName	string	The name of the workspace.
workspaceWaitTime		The total time a job step waited for workspace availability. On a job, this is the sum of time all job steps waited for workspace availability.
zoneId	number	The Commander-generated ID for this zone.

Returned element	Type	Description/Value
zoneName	string	The name of the zone.

Access Control

[Privileges](#)

[Users and groups](#)

[Special users and groups](#)

[Access Control Lists \(ACLs\)](#)

[Inheritance](#)

[System objects](#)

[Access control and jobs](#)

[Examples for increased security](#)

Overview

ElectricCommander provides a comprehensive mechanism to control how individuals use the system.

- Users must log in to view information or perform operations.
- After login, system access is limited based on:
 - user name
 - the groups to which that user belongs
 - permissions specified for various ElectricCommander objects

Access Control is the Commander functionality that provides security for all system objects. **After** you are familiar with the following information describing the Access Control system, review the two examples, Basic ACL Setup and Team ACL Setup (at the end of this help topic), which may provide guidance or insight for how you might setup enhanced Commander security at your site.

Privileges

ElectricCommander supports four privilege types for each object:

- **Read** - Allows object contents to be viewed.
- **Modify** - Allows object contents (but not its permissions) to be changed.
- **Execute** - If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.
- **Change Permissions** - Allows object permissions to be modified.

Users and Groups

ElectricCommander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository:

- Both user and group information is retrieved from the repository
- Local users and groups can be defined within ElectricCommander

Note: To view user and group information, and to modify local user and group information, click the Administration tab and select either the Users or Groups subtab. External account information **cannot** be modified from ElectricCommander.

If the same user exists in multiple sources, only the highest priority name is used. A priority order is defined among external repositories, but local names have the highest priority. Thus, if you define a local user with the same name as an LDAP user, you will effectively mask the user in the LDAP account.

For local user accounts, only local groups are considered—group information from external repositories is not used. For accounts from a particular repository, groups from that repository are used along with local groups, but groups in other repositories are not considered.

In other words, you can define local groups in ElectricCommander to supplement groups defined in external repositories. When you view information about users in Commander, only relevant groups are shown. For example, when you view group information for a local user, only local groups are displayed.

Note: Groups are managed by name only, without regard to source. If a particular group name exists in different repositories, there is no way to distinguish between these groups inside ElectricCommander. Access given to one group is the same for any other group with the same name.

Special Users and Groups

The **admin** local user has special significance. If you are logged in as "admin," you automatically have all privileges on all objects, regardless of any other system settings. This privilege set is a fallback mechanism in the event too many privileges get removed for an object, leaving it unusable.

The admin account cannot be deleted.

If the admin account is missing, ElectricCommander will recreate the account the next time it starts up with password "changeme."

The **Everyone** group is predefined by ElectricCommander and cannot be deleted. Every user is automatically a member of the Everyone group.

For each project, Commander uses the project name and automatically defines a user associated with that project, called the project principal. For example, the **project principal** for a project named "nightly builds" is "project: nightly builds" (notice that there are two spaces in this name)—this principal is used for jobs running under the project, as described in the "[Access control and jobs](#)" section later in this help topic.

Access Control Lists (ACLs) - allow and deny

Each ElectricCommander object, such as a project, procedure, job, property sheet, workspace, schedule, or resource contains an access control list. Individual properties do not have their own access control, but instead

use access control lists from their parent containers.

To view the access control list for an object, go to the web page displaying the object and click the **Access Control** link. An access control list can contain any number of entries— each entry names a particular user or group and indicates *allow*, *deny*, or is blank for each of the four privileges.

To determine whether a user can perform an operation on a particular object, ElectricCommander determines which of the four privileges is required for that operation, then searches all access control entries that refer to that user or groups containing that user. To be allowed access, at least one of the matching entries must specify "allow" and none of the entries can be "deny." A "deny" entry overrides an "allow" entry in the same ACL.

Inheritance

If an object's access control list does not indicate what to do for the user who is trying to access it, ElectricCommander looks in the access control list for the object containing the target object, then its parent, and so on up to a top-level object covering the entire server. This mechanism is called *inheritance*.

For example, a nested property sheet on a procedure inherits access control information from its parent property sheet, the procedure, the project containing the procedure, and the server. When you view the object access control list, you also see inherited access control lists so you can trace the object's inheritance chain.

When ElectricCommander performs an access check on an object, it begins with the access control list for that object.

- If a "deny" entry matches an entry in that list, access is denied.
- If there is no "deny" entry, but a matching "allow" entry is found, access is allowed.
- If there is no matching entry that specifies either "allow" or "deny," ElectricCommander moves to the next access control list in the inheritance chain and repeats the process.
- A matching entry in a lower-level access control list takes priority over entries in higher-level access control lists. If no matching entry in any list is found, access is denied.

Inheritance allows you to control access to a collection of objects because you can make changes in just one place. When new objects are created, their access control lists are empty, so all access control for the entire system is determined by the server-level list.

- Typically, when a new project is created, you define an access control list for the project and allow everything in the project to inherit from the project.
- If you do not define additional access control information in project components, all project access is determined by the project's access control list.

In some situations you may want tight control over object access, so changes in parent objects do not affect access to the object or its children. To achieve this, you can "break inheritance," which means a particular object does not inherit from its ancestors. The Commander web interface allows you to break and restore inheritance for any object on which you can change permissions. However, ***if you break an object's inheritance, it affects all children of that object.***

Additional Information about "Deny"

If a user matches a deny rule, however indirectly, the user is denied access. Even if a specific ACL entry granting (for example) read access by user name exists, a less specific deny ACL entry overrides that specific

rule. Therefore, avoid "deny" rules if at all possible (especially to groups), or you may experience some unexpected permissions issues.

You can, however, enforce an implicit "deny" by allowing an object to fall off the end of the inheritance chain. This practice allows denial behavior without the risks associated with an explicit ACL entry to deny access.

IMPORTANT: If you break inheritance on an object with an empty access control list, the object will become completely inaccessible. You will not be able to restore inheritance because you no longer have the right to change permissions on that object. If this happens, you must contact your system administrator who can login as admin and restore inheritance.

Consider Commander principals (actors) from two perspectives:

- A running job or workflow runs using the project as the identity, so if a job requires special permissions you must use the project as the principal in the ACL entry.
- Users and groups come into play when humans manipulate the GUI or execute ectool commands from the command line (or via a shell script) -- basically anywhere that a script or human needs to log in to Commander. Having logged in, a group and id exist, which can be used in ACL entries.

For example, consider a property that will store a build number. You can set an ACL that breaks inheritance (implicit deny), and then add an ACL entry granting permissions for the project to be able to increment the property value. This prevents humans from manipulating the build number but permits a running job to increment the build number at will.

Notes:

- When it's practical, a job will run with multiple identities—the project, as well as the user that launched the job. You cannot, however rely on this behavior for security because jobs run from a schedule (such as CI jobs) and jobs started by a workflow do not have a user principal, just the project principals.
- A job can have multiple project principals because it "accumulates" new project principals when it calls into another project's subprocedure (and then releases the other project's identity when the subprocedure returns). This makes it possible to create trusted libraries.

System Objects

A few special *system objects* contain access control lists related to overall ElectricCommander system administration. These access control lists are available from the Administration page. The system objects are:

- **Server** - an ElectricCommander system top-level object. Every other object in the system is contained in the server object and inherits access control information from the server object unless inheritance is broken.
- **Administration** - *Read* permission allows access to the `getStatus`, `getDatabaseConfiguration[s]`, `getEmailConfig[s]`, and `export(global)` API functions.
Modify permission allows access to the `shutdown`, `setDatabaseConfiguration`, `createEmailConfig`, `deleteEmailConfig`, `modifyEmailConfig`, and `import(global)` API functions.
- **Artifacts** - *Read* permission allows access to the `getArtifact(s)` API functions.
Modify permissions allows access to `createArtifact` and `deleteArtifact` functions.

- **Directory - Read** permission allows access to the `getUsers`, `getGroups`, and `getDirectoryProviders` API functions.
Modify permission allows access to the `createUser`, `createGroup`, `deleteUser`, `deleteGroup`, `createDirectoryProvider`, `modifyDirectoryProvider`, `deleteDirectoryProvider`, `testDirectoryProvider`, and `moveDirectoryProvider` API functions.
- **Email Configurations - Modify** permission allows access to the `createEmailConfig` and `deleteEmailConfig` API functions.
- **Force Abort - Execute** permission controls access to the `--force` flag on `abortJob`. By default, the ACL is created with Everyone: execute permission in addition to inheriting from the "Server". To force abort a job, the user must have execute permission on the job as well as execute permission on the `forceAbort` ACL.
- **Logging - Modify** permission allows access to the `logMessage` API function.
- **Licensing - Read** permission allows access to the `getLicense[s]` API functions.
Modify permission allows access to the `importLicenseData` and `deleteLicense` API functions.
Execute permission allows access to the `preemptLicense` API function.
- **Plugins - Modify** permission allows access to the `createPlugin`, `deletePlugin`, `installPlugin`, `uninstallPlugin`, `promotePlugin` API functions, and the `modifyPlugin` API function requires modify permission on the target plugin. For `getPlugin`, *Read* permission is required on the target plugin.
- **Priority - Execute** permission allows the user who launches a procedure (using the `runProcedure` API function) to raise the priority of the job.
- **Projects - Modify** permission allows access to the `createProject` and `deleteProject` API functions.
- **Repositories - Read** permission allows access to the `getRepository` API function.
Modify permission allows access to the `createRepository`, `deleteRepository`, `modifyRepository`, and `moveRepository` API functions.
- **Resources - Modify** permission allows access to the `createResource` and `deleteResource` API functions.
- **Session - Execute** permission allows access to the `login` API function.
- **Workspaces - Modify** permission allows access to the `createWorkspace` and `deleteWorkspace` API functions.
- **ZoneAndGateways - Modify** permission allows access to the `createZone` and `deleteZone` API functions. *Modify* permission also allows access to the `deleteResource` API function when the resource belongs to a Gateway.

Access Control and Jobs

When a job executes, it usually needs to access objects in ElectricCommander. For example, a job step command may refer to a parameter value, which is a property associated with the job object, or a step may invoke `ectool` to modify properties or any other Commander state. This process leads to questions:

- ***Under which user name does the job execute?***

Procedures always run under the project principal user ID for the project that contains the procedure. If a procedure invokes a subprocedure in another project, that subprocedure will run under its own project's project principal and the project principal of its calling procedure. When a procedure is running under

multiple project principals, its steps will perform any operations for which any one of its project principals allow.

- ***How does ElectricCommander initialize job permissions when the job starts?***

This question pertains to job object permissions. When a job starts, ElectricCommander sets full access control entries on the job for the project principal and the user who launched the job— assuming the job was launched by a user and not a schedule.

- ***What permissions are needed to abort a job?***

Aborting a job requires *execute* permission on the job. If a job is launched by a user, that user is given all privileges on the job. If a job is launched by a schedule, the schedule's execute privileges are copied to the job.

The access control system determines whether jobs can be executed or not.

- For a user to run a job without creating a schedule, the user must have *execute* permission on the top-level procedure being executed.
- To create a schedule to run a procedure, a user must have *modify* permission for the project containing the schedule. After a schedule is created, no additional permissions are required to start jobs under the auspices of that schedule.

Examples for Increased Security

Initially, the *Everyone* group had Read and Execute permission on all Commander objects by default. The following examples illustrate how you might customize Commander default Access Control settings for your organization.

Caution: Make sure you are familiar with the Access Control system before making changes. Incorrectly setting ACLs can severely impact Commander behavior.

Abbreviations used in the examples

A = Allow

I = Inherit (some places referred to as "Don't Care")

D = Deny

Change Permission = Change

Example 1 - Basic ACL Setup

This example illustrates how you might change Access Control defaults to set up a basic level of security. The easiest way to manage ACLs is by using different Commander groups that allow users to perform specific roles.

For our example, we will create and use the groups called EC-administrator and EC-designer, and we will modify the default Everyone group. (When setting up your groups, you would choose any group names that suit your organization, but note that the "EC-" prefix is reserved for Electric Cloud use only.) This is how we want to define our groups:

- EC-administrator group
This group has most of the power and abilities of the 'admin' user login, but allows for more flexibility.

- EC-designer group
This role is more significant. Users in this group can create and modify most Commander objects.
- Everyone group
This predefined group is used for people who only need to run Commander procedures, and have no need to create or modify them.

As you begin to modify the Access Control system, note that the system has some project principal ACEs that should remain in place for correct system operation.

Server ACLs

The Server is the foundation for almost all ACL checks. Generally, every other object in the system inherits these privileges. For this basic ACL configuration we want to set Server ACLs this way:

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Custom Server Properties ACLs

An often overlooked ACL setting is Server Property Sheet. Allow the Everyone group Read permission on this object.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

System Object ACLs

There are 14 categories of System objects with ACLs.

1 - Administration – allow the EC-designer group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	I	I	I

2 - Artifacts – allow the EC-designer group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	A	I	I
group	Everyone	A	I	I	I

3 - Directory – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	I	I	I

4 - Email Configurations – allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

5 - Force Abort - no changes needed for this category.

6 - Logging - no changes needed for this category.

7 - Licensing - no changes needed for this category.

8 - Plugins - allow the Everyone group Read privilege.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

9 - Priority - no changes needed for this category.

10 - Projects - allow the Everyone group Read and Execute privileges, and the EC-designer group everything except Change privilege.

Type	Name	Read	Modify	Execute	Change
group	EC-designers	A	A	A	I
group	Everyone	A	I	A	I

11 - Repositories - allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

12 - Resources - allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

13 - Session - allow the Everyone group Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	I	I	A	I

14 - Workspaces - allow the Everyone group to have Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Note: Sites that want a slightly more open system might consider granting the "EC-designers" group Modify access to Resources and Workspaces.

Plugin Project ACLs

Generally, you will want to allow the Everyone group Read privileges on every plugin you expect to use. (Each plugin has its own name, which is the project name also.)

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

Project ACLs

Next are special Electric Cloud projects, Default, EC-Utilities and EC-Examples.

Projects: Default, EC-Utilities, EC-Examples

Use caution when setting privileges on the EC-Utilities project. These are powerful procedures and some have their own ACL settings. Remember the 'admin' user will always be able to use these utilities. Allowing the EC-administrator group full privileges accomplishes the same goal for users in that group. We allow the Everyone group Read only privileges. This is a unique project because it does not inherit ACLs from anywhere else (for example, Server or Projects).

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	A	I	I	I

Example 2 - Team ACL Setup

Some organizations may want to have two or more different and separate teams use Commander at the same time. In this example, people on one team have little knowledge about the other team who uses Commander and no direct access to any objects used by the other team. Our "Team" example describes how to modify default Commander Access Control settings to create a multi team security configuration using ACLs.

To divide your Commander system for use by separate teams, you will want to create one or more separate Projects for each team. You may decide to share some projects across teams.

Create your teams (groups). For our example, we will begin with two teams, T1 and T2. Each team has a designer group and a user group. Using groups allows the flexibility to have a designer from one group work as a user in another group.

- Create the EC-administrator group
This group has most of the power and abilities of the 'admin' user login, but allows for more flexibility. Assigning people to the EC-administrator group allows better tracking of Commander administrative activity.
- Create two designer groups: T1-designer and T2-designer
Members of the designer groups will have the ability to create and modify procedures and other Commander objects for their team. A member of the designer group from one team will not be able to view or use the objects of the other team.
- Create two user groups: T1-user and T2-user
Members of the user groups will have the ability to run procedures that belong to their team. A member of the user group from one team will not be able to view or use objects that belong to the other team, who are not in any other group. This setup allows other users to access Commander without exposing the work of any of the teams. A person not in a group for either team will not be able to see objects of either of the teams.

As you begin to modify the Access Control system, note that the system has some project principal ACEs that should remain in place for correct system operation.

Server ACLs

The Server is the foundation for almost all ACL checks. Almost all objects in the system inherit these privileges. For this basic Team configuration, we will set the Server ACLs this way:

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Custom Server Properties ACLs

An often overlooked ACL setting is Server Property Sheet. Allow the Everyone group Read permission on this object.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

System Object ACLs

There are 14 categories of System objects with ACLs.

1 - Administration – allow both designer groups Read privileges.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	I	I
group	T2-designer	A	I	I	I

2 - Artifacts – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

3 - Directory – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

4 - Email Configurations – allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

5 - Force Abort - no changes needed for this category.

6 - Logging - no changes needed for this category.

7 - Licensing - no changes needed for this category.

8 - Plugins - allow the Everyone group Read privilege. Generally, privileges set here will be inherited by all plugins.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

9 - Priority - no changes needed for this category.

10 - Projects - no changes needed for this category. Privileges for projects are now handled on a case by case basis.

11 - Repositories – allow the Everyone group Read privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

12 - Resources - allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

13 - Session - allow the Everyone group Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	I	I	A	I

14 - Workspaces - allow the Everyone group Read and Execute privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Plugin Project ACLs

Generally, you will want to allow the Everyone group Read privileges on every plugin you expect to use. (Each plugin has its own name, which is also the project name.) Because you may want to use many plugins, you might want to use a script.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

ACLs for Existing Projects

Next are a few Electric Cloud projects, and two sample projects that can be seen and used by their respective team members only.

Project: EC-Examples

These are nice examples and we want to grant Everyone the ability to see and run them.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Project: Default

Inherited settings will allow the EC-administrator group full privileges.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

Project: EC-Utilities

Use caution when setting privileges on the EC-Utilities project. These are powerful procedures and some have their own ACL settings.

Remember the 'admin' user will always be able to use these utilities. Allowing the EC-administrator group full privileges accomplishes the same goal for users in that group. We allow the Everyone group Read only

privileges. This is a unique project because it does not inherit ACLs from anywhere else (for example, Server or Projects).

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	A	I	I	I

Project: Electric Cloud

This project has one procedure and we want to grant everyone the ability to see it.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	I	I

ACLs for New Projects

Now we introduce the concept of ACL settings as they would be in a multiple team environment. For this example, we assume you have five projects being used by two teams in the following way.

- Project-A used by Team1
- Project-B used by Team1
- Project-C used by Team2
- Project-D used by Team2
- Project-E used by Team1 and Team2

Project-A

This project is visible and usable only by Commander users who are members of either the T1-user group or the T1-designer group. Notice that the designer group receives full permissions.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A
group	T1-user	A	I	A	I

Project-B

This project is the same as Project-A.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A
group	T1-user	A	I	A	I

Project-C

This project is visible and usable only by Commander users who are members of either the T2-user group or the T2-designer group. Notice that the designer group receives full permissions.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Project-D

This project is the same as Project-C.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Project-E

This project is a superset that includes both teams.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	A	A	A
group	T1-user	A	I	A	I
group	T2-designer	A	A	A	A
group	T2-user	A	I	A	I

Enabling Same Team Subprocedures

To enable procedures in one project to use procedures in another project, one more step is necessary.

Add the project principals for both T1 projects to the T1-user group.

In this case, these are called: “project: Project-A” and “project: Project-B”.

Add the project principals for both T2 projects to the T2-user group.

In this case, these are called: “project: Project-C” and “project: Project-D”.

Optional: Restricting Resources and Workspaces by Team

Resource ACLs

Resource: local

Because each team will have dedicated resources, it is useful to keep one resource available for everyone to use.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

T1-resource

This resource is visible and usable only by Commander users who are members of either the T1-user group or the T1-designer group. This resource also needs Execute permission for the project itself to allow it to run from a schedule.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	A	I
group	T1-user	A	I	A	I

T2-resource

This resource is visible and usable only by Commander users who are members of either the T2-user group or the T2-designer group. This resource also needs Execute permission for the project itself to allow it to run from a schedule.

Type	Name	Read	Modify	Execute	Change
group	EC-administrator	A	A	A	A
group	Everyone	I	I	I	I

Workspace ACLs

Workspace: default

Because there will be dedicated workspaces for each team, it is useful to keep one workspace available for Everyone to use.

Type	Name	Read	Modify	Execute	Change
group	Everyone	A	I	A	I

T1-workspace

This workspace is visible and usable only by Commander users who are members of either the T1-user group or the T1-designer group.

Type	Name	Read	Modify	Execute	Change
group	T1-designer	A	I	A	I
group	T1-user	A	I	A	I

T2-workspace

This workspace is visible and usable only by Commander users who are members of either the T2-user group or the T2-designer group.

Type	Name	Read	Modify	Execute	Change
group	T2-designer	A	I	A	I
group	T2-user	A	I	A	I

Artifact Management

- Artifact objects
- Access Control
- Publishing artifact versions
- Enable Compression
- Include / Exclude patterns
- Retrieving artifact versions
- Supplying filters during a retrieve operation
- Dependent artifact versions
- Artifact cache
- Cleaning up repositories and caches

Overview

During the course of the development process, project components or "outputs" are produced for consumption by other projects or the main project to create a whole entity. Ultimately, applications are assembled by combining these various outputs (dependencies) and packaging them together into a single deployable software application.

Outputs can be libraries, scripts, graphics, and so on. This output is an **artifact** that can be consumed by other projects.

Every stage in the application lifecycle produces and consumes artifacts. Artifacts are critical to the *build-test-deploy* process.

Using artifacts can:

- improve performance across builds
- provide better reusability of components
- improve cross-team collaboration with greater traceability

For example, instead of each developer repeatedly downloading third-party packages from external sources, these components can be published and versioned as an artifact. A developer then simply retrieves a specific artifact version from a local repository, guaranteeing a consistent package from build to build.

Artifact objects

ElectricCommander has three objects to support Artifact Management functionality: **artifact**, **artifact version**, and **repository**. Similar to other Commander objects, each of these objects supports custom properties and

access control.

Artifact

An **artifact** is a top-level object containing artifact versions, a name template for *published* artifact versions, artifact specific properties, and access control entries to specify privileges.

If an artifact is deleted, all published artifact versions within that artifact will be deleted also.

You can create artifacts in several ways:

- Use the New Artifact web page
- Use the `createArtifact` API command with the command-line tool (ectool) or an ec-perl script
- An artifact is created when you publish an artifact version if the artifact did not already exist, and the user issuing the publish command has permission to create artifacts. See the "Access Control" and "Publishing Artifact Versions" sections below for more information.

To create an artifact, you supply a Group Id and Artifact Key and together, these create an artifact name in the form `groupId:artifactKey`. After Commander creates the artifact name from your supplied Group Id and Artifact Key specifications, you cannot modify the name.

Note: Plan your artifact Group Id's and Artifact Keys carefully to prevent the extra work of deleting a name and beginning your specifications again. The use of a `groupId` and `artifactKey` allow two "namespaces" for added flexibility in organizing artifacts within your company or organization.

Examples

- If your development team uses GWT for web development and JUnit for unit testing, you may want to create artifacts `"ThirdParty:GWT"` and `"ThirdParty:JUnit"`.
- Your organization may have two different development teams (OS and Apps) who want to use artifacts. In building each product, both development groups want to publish and use an "SDK" with their product. But each SDK is different and unique. One solution would be to create one artifact named `"os:SDK"` and another artifact named `"apps:SDK"`. By using different group Id's, each development team is free to name their artifacts however they choose without worrying about name collisions with artifacts produced by the other team.

To reference an artifact, you can use the `groupId` and `artifactKey` combination or simply use the `artifactName`. The property path `/artifacts/<artifactName>` allows you to access any properties of a given artifact.

Artifact version

An **artifact version** is a collection of 0 to N files that were published to an artifact repository. Artifact version metadata is stored in an `artifactVersion` object in the Commander server.

Tips for working with artifact versions:

- To group a collection of files (typically from build output in a Commander workspace) into an artifact version, specify the "from directory" containing those files.
- If you want to include files from multiple directories in your artifact, you need to specify a common root directory to include these directories when publishing your artifact.

- To restrict which files and subdirectories in the "from directory" to include in the artifact version, you can specify "include" and "exclude" patterns.
- Artifact versions are stored on an artifact repository server in either uncompressed `tar` archives or compressed `tar-gzip` archives.
- Artifact versions can be created using the `publishArtifactVersion` API, which includes creating the object in the Commander server and publishing the artifact version to the artifact repository.
 - **Note:** Interactively creating an artifact version using the Commander web UI is not supported at this time.
- When you retrieve an artifact version, the original directory structure of the directory where files were published "from" is preserved.
- When you retrieve/publish artifacts, directories containing symlinks are always resolved; they are not preserved.

Artifact versions are referenced by *Group Id*, *Artifact Key* (or artifact name) and a version string.

- The artifact version's name is set based on the name template on the "owning" artifact. By default, the name template is in the form `groupId:artifactKey:version`.
- Each artifact version name must be unique within a Commander server installation.

The property path `/artifactVersions/<artifactVersionName>` allows access to any properties for a particular artifact version. Because you may not know the artifact version name (because of the name template), this path supports the form `<groupId>:<artifactKey>:<version>` as a substitute for the name of the artifact version.

All Commander API's that take an `artifactVersionName` argument also accept this alternate form as a substitute, similar to how other API's that accept a `jobId` argument accept the job name as a substitute.

Version strings

A version string must be provided when publishing an artifact version.

For example:

`5.8.8-EN-55842` or `5.8.8.55842-EN` – In either form, the version string is interpreted as:

Major version number:	5
Minor version number:	8
Patch level number:	8
Qualifier:	EN
Build or Job Id:	55842

You must provide separator punctuation. When interpreting (parsing) the first version string form, Commander interprets the text after the first hyphen as the build number if this string contains numeric characters only. Otherwise, the string is interpreted as the `qualifier`.

Version string examples and how Commander interprets them:

	Input (version string)	Major	Minor	Patch	Qualifier	Build Number
1	2.0.3-44834	2	0	3	<empty/null>	44834
2	3.8.4-RC1	3	8	4	RC1	0
3	3.8.4-RC1-36	3	8	4	RC1	36
4	3.8.4.36	3	8	4	<empty/null>	36
5	1-36	1	0	0	<empty/null>	36
6	\$_[jobId]-DE	\$_[jobId]	0	0	DE	0
7	3.8.4.36-RC1	3	8	4	RC1	36

Table explanations:

- Row 5 shows that you are not required to specify all three of `<major, minor, patch>` when publishing an artifact version if your organization's versioning conventions do not use all three fields. However, there is a caveat with this level of flexibility. You can create two artifact versions in the system with equivalent versions.
Version string "1-36" is equivalent to "1.0-36" but they are not the same so you can publish two artifact versions with these version strings successfully. When retrieving an artifact version, in some cases it could be unclear as to exactly which one you will get. For this reason, Electric Cloud recommends that you adopt a convention and abide by it for a particular artifact.
- Row 6 - A Job ID property was used as part of the version string. When expanded, the Job ID becomes a numeric string. Therefore, Commander interprets the Job ID as the "major version" number of the `major.minor.patch` string. DE becomes the qualifier, and no build number is set.

The artifact version object exposes components of the version string by using these intrinsic properties:

- version** - the version component is the combination of `major.minor.patch-qualifier-buildNumber`. A "version" component does not need to include all of the following intrinsic properties. See the table above.
- majorMinorPatch** - this is the `major.minor.patch` version component as specified in the version string. For example, if you specified a version string of "1-36, the `majorMinorPatch` component would be "1", not "1.0.0", which makes it possible for you to meet your naming conventions by reconstituting the version string a different way in the artifact version name template.
- qualifier** - the qualifier component of the version .
- buildNumber** - the build number component of the version.

The artifactVersionState property

Artifact versions have an `artifactVersionState` property whose value can be one of the following:

`publishing` - The artifact version is currently being published and is not available for retrieval.

`available` - The artifact version is available for retrieval if needed.

`unavailable` - The artifact version is not available for retrieval while in this state.

You can manipulate the state between available and unavailable on the Edit Artifact Version web page. This action can be useful if an artifact version seems to be corrupt in some way, but you want to investigate before potentially deleting it from the system.

By making the artifact version unavailable, retrievers can acquire an older (potentially more stable) artifact version while you look into the problem. If the artifact version is good, you can simply make it available again.

Repository

The artifact repository is a machine where artifact versions are stored. The repository server is configured to store artifact versions in a directory referred to as the repository backingstore.

By default, the backingstore is the `<datadir>/repository-data` directory in the repository installation. This default setting can be changed by running `ecconfigure --repositoryStorageDirectory` on the repository server. The repository server listens on port 8200 for HTTPS requests to publish and retrieve artifact versions.

Connection information is stored in the repository object on the Commander server.

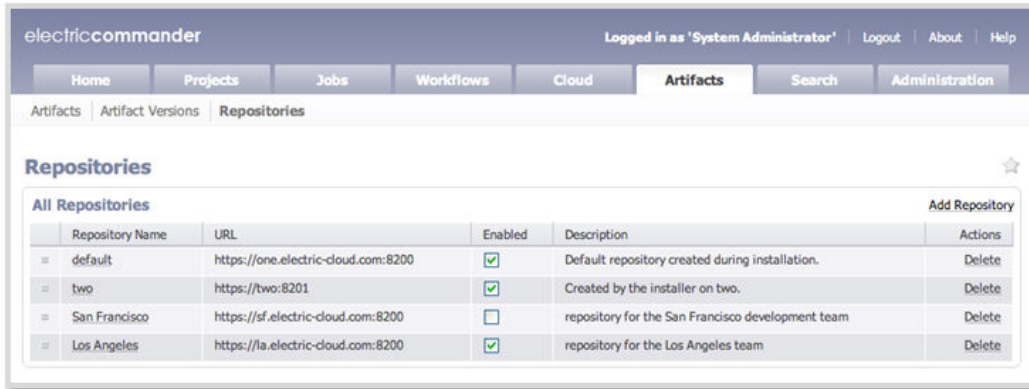
You can create a repository objects in several ways:

- Use the New Repository web page in the Commander UI
- Use `createRepository` API
- Repository objects can be created automatically during the Commander installation.

When you are installing a repository *server*, the installer creates a corresponding repository *object* on your Commander server. If you are installing a repository *server* on the same machine as your Commander server, a repository *object* named "default" is automatically created. If you are installing a repository *server* on a different machine than your Commander server, the installer will prompt you for information it uses to create the corresponding repository *object*.

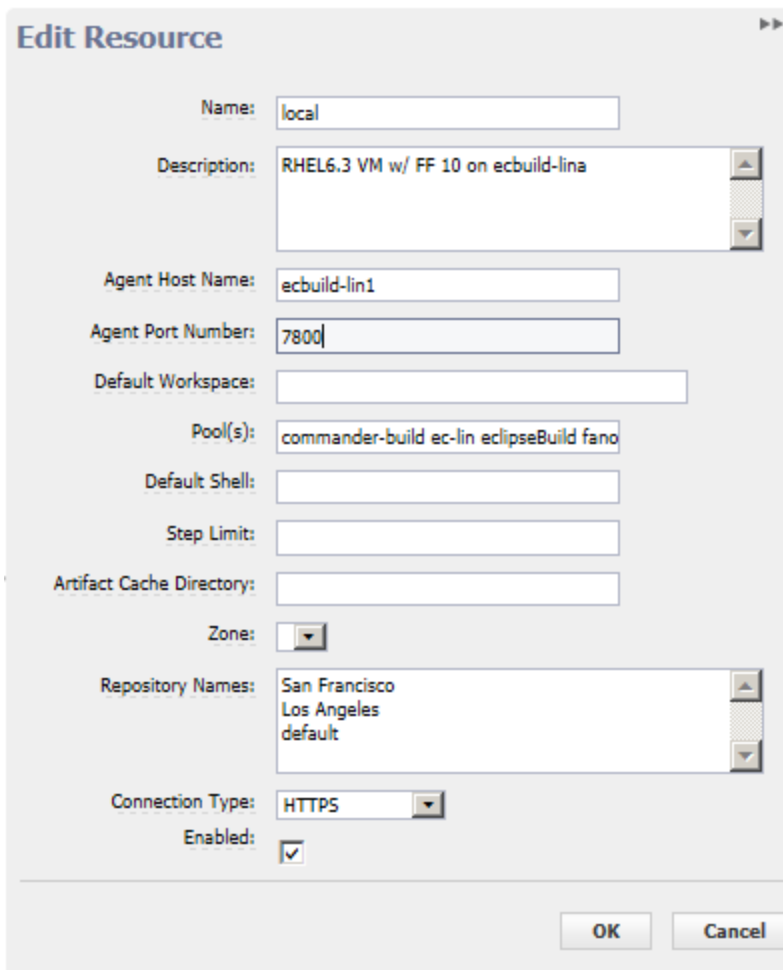
Note: When installing a repository server, Electric Cloud recommends installing an agent also to ease maintenance tasks such as clearing out stale artifact versions from the repository backingstore.

Click the Artifacts tab, then the Repositories subtab to see your list of created repositories.



You can re-order the repository search order (for retrieving artifact versions) by using the mouse to grab and drag the icon in the left column to the row position you prefer, effectively re-ordering the repository list.

For distributed environments, where the preferred repository search order varies depending on which resource is performing the retrieval, you can specify a preference order on the Edit Resource panel. See the illustration below.

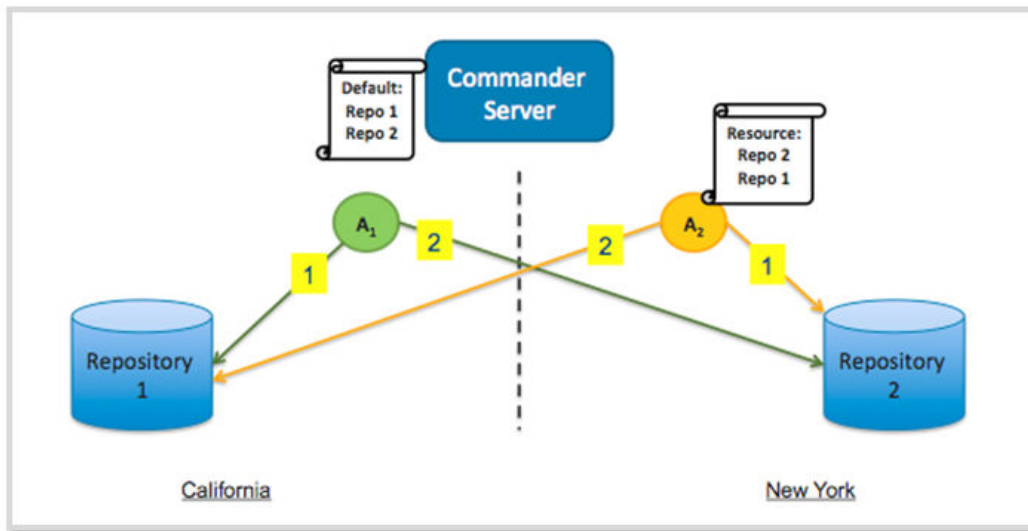


To go to the Edit Resource panel; from the Repositories page, select the Cloud tab, which opens to the Resources page, then select a resource name

Enter one repository name per line in the Repository Names field.

In the following example, a resource in California (A1) uses the "default" repository search order (that is, the order on the Repositories page). A1 searches for artifact versions by first requesting an artifact version from Repository 1. If the artifact version requested is not found, it will then request the artifact version from Repository 2.

The resource in New York (A2) has a specific search order specified where it searches for artifacts from Repository 2, first, then searches Repository 1.



Using multiple repositories

Commander supports multiple repositories, however you cannot share a repository between multiple Commander servers. Multiple repository servers may be required in your organization because of business or organizational reasons, government and compliance requirements, or for performance and data protection.

Examples:

Business or Organizational Structure

Within a company, two different development teams want to have artifact repositories containing artifacts and artifact versions specific to their product. A repository server is created for each team. The two teams publish artifact version to their respective repositories.

Performance

A company has multiple development sites. Instead of having a single repository server, which would create significant network performance issues from retrieving artifact versions during the build process, they create a repository at each development location and have artifact versions synchronized between

the repository servers. In this way, when artifact versions are retrieved for use by a build process, they are retrieved from a local repository.

Data Protection and disaster recovery

A company has a single Commander server with two artifact repository server machines. The contents of the repository servers are synchronized and replicated between both machines. In the event repository 1 is unavailable during a retrieve, repository 2 is searched for the requested artifact retrieval.

The repositoryDisabled intrinsic property

The repository object has a `repositoryDisabled` property that dictates whether or not artifacts can be published or retrieved from a particular repository. If you take a repository offline for maintenance, you can disable the repository in the Commander server rather than shutting down the repository machine or the service. Steps that attempt to publish to this repository will fail, and steps that attempt to retrieve artifact versions will skip this repository during the search for available repositories.

Access control

You can apply access control permissions to artifacts and artifact versions to control who can create an artifact versus who can publish an artifact version versus who can read or retrieve an artifact version.

Action	Required permission	Object
Create an artifact	Modify	artifacts system object
Read artifact metadata	Read	the relevant artifact
Modify an artifact	Modify	the relevant artifact
Delete an artifact, which deletes all artifact versions within the artifact	Modify	artifacts system object
Publish an artifact version	Modify	the artifact into which a new version is being published

Action	Required permission	Object
Read artifact version metadata (required for retrieval)	Read	the relevant artifact version
Modify an artifact version	Modify	the relevant artifact version
Delete an artifact version	Modify	the artifact that owns the artifact version

Note: The Execute privilege does not take part in artifact or artifact version access control.

"Out-of-the-box," an `<Everyone, modify>` access control entry is available that allows anyone to publish an artifact version to any artifact, and allows anyone to create an artifact. You might consider removing this access control entry and setting up tighter control for individual artifacts. See the EC-Security plugin for help with this effort.

For more information about access control for artifact management, see the [Access Control](#) help topic.

The following scenarios provide ideas for how you might set up access control for artifact management.

Control who can create new artifacts and publish artifact versions

Commander administrator, Adam, received communication from development manager, Max, requesting that only he should be able to create artifacts in Commander -- developers of Commander procedures under him should only be able to create steps to publish artifact versions under pre-defined artifacts.

His reasoning is that it is too easy for a developer to introduce a typo during publish (which does auto-create the artifact if the user has the appropriate privilege and the artifact doesn't exist).

Adam accommodates the request by giving Max the modify privilege on the artifacts system object and removing the `<Everyone, modify>` access control entry. Max then creates artifacts and assigns modify privileges to those artifacts for his development group, effectively allowing them to publish artifact versions.

At run-time, a step belonging to some project actually performs the publish, so Max adds project principal access control entries for the relevant projects as well, providing modify privileges.

Control which projects can retrieve artifact versions at run-time

Acme Corporation's router product has a software stack consisting of OS, PlatformLibraries, and Apps. Each of these has a Commander project with several procedures that work together to build the relevant

component. PlatformLibraries builds against some of the C header files and libraries produced by the OS build. Similarly, Apps builds against libraries produced by PlatformLibraries.

The lead developer for the OS component, creates a `Router:OSLib` artifact and gives the PlatformLibraries project principal read privileges so that project can retrieve any OSLib artifact version it needs to build PlatformLibraries. Similarly, Paul from the PlatformLibraries team creates a `Router:PlatformLib` artifact and gives the Apps project principal read privileges.

Group 1 publishes an artifact version and wants only Group 2 to be able to retrieve it

An organization wants to create artifacts for each of the third-party packages used by developers. Among the tools shared is the GWT package. The build manager chooses to create an artifact called `3rd-party-PKGS:GWT`. Because of licensing concerns, the build team must control or regulate who has access to the GWT and who can publish artifact versions used within the company's product.

In this case, the build team will publish a GWT artifact version and test it. During the test and approval cycles, only the build team has access to read and modify the artifact or publish new artifact versions. After a version has passed internal testing and approvals, the build team then grants read access to each of the developer groups.

Publishing artifact versions

An artifact version is published by using a procedure step or the *ectool* / *ec-perl* command-line interface. In most cases, using the Publish Artifact Version step is sufficient.

The following screen illustrates the parameters section for creating the Publish Artifact Version step:

Parameters

Artifact: Required

Version: Required

Repository: Required

Enable Compression? ☒

From Location:

+ Add Include Pattern

+ Add Exclude Pattern

Dependent Artifact Version(s):

perl.bin:[6]	Remove
perl.lib:[6]	Remove
perl.man:[2]	Remove

+ Add Dependent Artifact Version

Enable compression

When publishing an artifact, consider whether or not to enable compression. Using compression reduces transfer time during publish. 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.

Artifact versions that contain installers, jars, audio, and video are almost certainly **not** compressible because these file types are already compressed, so the cost of compressing outweighs the (near zero) benefit of

reducing transfer time. Another consideration: Artifact versions are stored in the same format as they are transferred, so highly compressible artifact versions use less space in the repository backingstore.

Include / Exclude patterns

When publishing an artifact, you can choose which files to include by using include and exclude patterns. File patterns are expressed as relative paths under the From Directory. Pattern syntax and behavior is the same as Ant and uses the following wildcard characters:

? - matches a single character

* - matches any number of characters, but only at a single directory level

** - matches any number of directory levels

Examples:

Use *.jar to match any .jar file in the top-level directory.

Use */*.jar to match any .jar file in any child directory.

Use **/*.jar to match any .jar file at any level.

Retrieving artifact versions

An artifact version is retrieved using a procedure step, or the `retrieveArtifactVersion` API call, used by the `ectool/ec-perl` command-line tools. This API call returns the most current artifact version that meets your criteria, per the algorithm described below.

Typically, you specify a version number range. Artifact version ranges are specified using interval notation. Brackets `[]` indicate versions to include and parentheses `()` indicate versions to exclude.

For example, the following artifact versions are published:

```
foobar:test:3.0.1-3645
foobar:test:1.2.3-2139
foobar:test:2.0.0-4000
foobar:test:2.0.0-DE-3395
foobar:test:2.0.0-DE-2445
foobar:test:2.0.0-DE
foobar:test:3.0.0-3539
foobar:test:4.0.0-5584
```

You want to retrieve the most current artifact version for the `foobar:test` artifact. To do this, you do not need to provide a version or version range, but instead enter retrieve parameters in the Retrieve Artifact Version step page as follows:

Parameters

Artifact: Required

Version:

☒ Latest

☐ Exact:

☐ Range:

Minimum: ☐ Inclusive?

Maximum: ☐ Inclusive?

Retrieve to directory: ☐ Overwrite:

Retrieved Artifact Location Property:

+ Add Filter

The equivalent command using ectool is:

```
ectool retrieveArtifactVersions --artifactName foobar:test
```

and with ec-perl, the syntax is:

```
my $cmdr = newElectricCommander();
$cmdr->retrieveArtifactVersions({artifactName => "foobar:test"});
```

Version numbers for two artifact versions (Artifact Version 1 and Artifact Version 2) are compared in the server as follows:

Compare `major.minor.patch`. If they are equal,

Compare `qualifiers`. If they are equal,

Compare `buildNumbers`.

Note: This comparison algorithm does not take the time of publish into account. "Current" means the artifact version with the greatest version per the above algorithm.

So the above list of artifact versions would be sorted (most current to least as follows:

```
foobar:test:4.0.0-5584
foobar:test:3.0.1-3645
foobar:test:3.0.0-3539
foobar:test:2.0.0-DE-3395
foobar:test:2.0.0-DE-2445
foobar:test:2.0.0-DE
foobar:test:2.0.0-4000
foobar:test:1.2.3-2139
```

If you want to retrieve the latest 3.0.0 artifact version, set maximum version to 3.0.1 "exclusive".

The parameters entered in the retrieve step would look like this:

Parameters

Artifact: Required

Version: ☐ Latest ☐ Exact: ☒ Range:

Minimum: ☐ Inclusive?

Maximum: ☐ Inclusive?

Retrieve to directory: ☐ Overwrite:

Retrieved Artifact Location Property:

[+ Add Filter](#)

Setting this constraint ensures the entire "version string" is properly evaluated. In other words, you want to ensure all artifact versions with 3.0.0, including qualifiers and build numbers, are included in the evaluation.

The equivalent command using ectool is:

```
ectool retrieveArtifactVersions --artifactName foobar:test
--versionRange "(,3.0.1)"
```

And with ec-perl the syntax is:

```
my $cmdr = newElectricCommander();
$cmdr->retrieveArtifactVersions({artifactName => "foobar:test",
    versionRange => "(,3.0.1)"});
```

Note the use of parentheses to exclude version "3.0.1".

Generally, while in development, you will want to specify a minimum artifact version and take the latest version published, then when an artifact has reached a "released" state, indicate a specific version.

Supplying filters during a retrieve operation

At times, a company might want to distinguish between artifact versions not just by version range but also by some other property of the artifact version.

Examples are:

- Specifying a qualifier in the version string when publishing.
For example, you might want to publish an artifact version for your product's English or German variants. You could publish the English version as 2.0.4-EN-55497 and the German version as 2.0.4-DE-55497.
- Setting a custom property on the artifact version.
For example, after your QA team has tested an artifact version, they could show approval by setting a `qaApproval` custom property, including the name of the QA engineer.

For either case, filters can be applied when retrieving an artifact version. To retrieve the latest German version for the artifact, the parameters entered on the retrieve artifact custom step page may look like this:

Because `DE` was used as the qualifier, Commander will retrieve the latest artifact version for the `perl:pkg` artifact where the qualifier equals `DE`.

Only the following filter operators can be applied to the intrinsic property version:

`equals`

`greater than` (`greaterThan`, for use in the API)

`less than` (`lessThan`, for use in the API)

`greater than or equals` (`greaterOrEqual`, for use in the API)

`less than or equals` (`lessOrEqual`, for use in the API)

"equals" is special because it does a string comparison on the version string of the artifact version, while the other operators compare the interpretation of the version string as described earlier.

All filter operators can be applied to version string components: `majorMinorPatch`, `qualifier`, or `buildNumber`. To find artifact versions with no qualifier, specify a filter on "qualifier" with operator "is not set" (or `isNull` in the API).

So for the QA approval case in a previous example, if a procedure is interested in an artifact version that was approved by QA, the retrieve step would specify a filter for `"qaApproval is set"`.

ectool does not currently support the filters argument, but you can write a Perl script as follows:

```
my $cmdr = new ElectricCommander();

$cmdr->retrieveArtifactVersions({artifactName => "perl:pkg",
    filters => {propertyName => "qaApprover",
        operator => "isNotNull"}});
```

Dependent artifact version(s)

A published artifact version can be dependent on a list of artifact versions. These dependent artifact versions are retrieved when the primary artifact version is retrieved and they are specified with a query syntax when publishing the primary artifact version.

For example:

An artifact version for the Commander product could include dependent artifact versions for the core product, the SDK, and online help:

- the latest `commander:Core` artifact version greater than or equal to version 3.5 (including 3.5)
- the most recent version of the `commander:SDK` artifacts
- the `commander:onlineHelp` artifact with a version greater than version 3.10 (excluding 3.10)

Because dependent artifact versions are evaluated during the retrieve process, specify them using the same syntax as you would for retrieving any artifact version, which is by including the group Id, the artifact key, and a specific artifact version or a version range.

Parameters for the "publish" custom step page would be similar to the following:

The screenshot shows a 'Parameters' form with the following fields and controls:

- Artifact:** Required
- Version:** Required
- Repository:** Required
- Enable Compression?** ☒
- From Directory:**
- Buttons:** [+ Add Include Pattern](#), [+ Add Exclude Pattern](#)
- Dependent Artifact Version(s):**
 - [Remove](#)
 - [Remove](#)
 - [Remove](#)
- Bottom Button:** [+ Add Dependent Artifact Version](#)

A published artifact version could have zero files, that is, no files are published as part of the artifact version, but it still could have a list of dependent artifact versions. An empty artifact version might be useful as modular convenience container for retrieving a group of artifact versions in downstream processes.

Artifact cache

Artifact versions are retrieved to an artifact cache directory before being consumed by a build process. Properly configured, the artifact cache directory can significantly improve efficiency as you work with artifacts.

The artifact cache directory is set in one of these ways:

- Supply information in the Artifact Cache Directory field on the Edit Resource page.
- Use the `modifyResource` API to set the `artifactCacheDirectory` intrinsic property.
- Set the `artifactCache` field in the `agent.conf` file.

Thus, each resource can have its own artifact cache location (directory). Alternatively, multiple resources as well as an entire site can share an artifact cache if the cache directory resides on a shared file server.

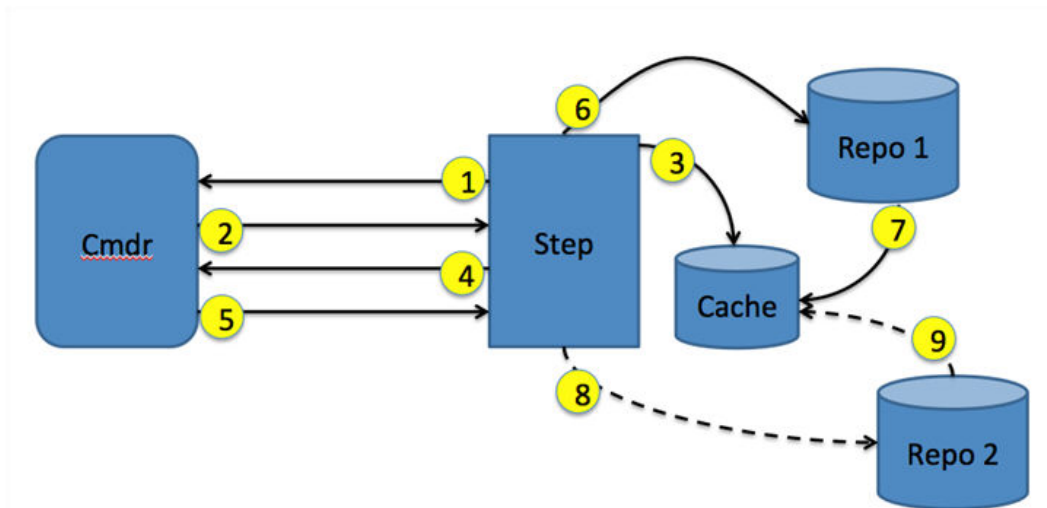
When a Commander step retrieves an artifact version, remember that often a version range or no version at all is specified. For example, to retrieve the latest Commander SDK artifact version, you could issue the following command:

```
ectool retrieveArtifactVersions --artifactName commander:SDK
```

No version is specified because we want the "latest" artifact version published.

In the following example, the step sends a request (1) to the Commander server.

- The Commander server replies (2) with a fully qualified artifact version name. In this case, it returns `commander:SDK:1.2.0.43552`.
- The step then looks in its artifact cache for the artifact version (3).
- If the artifact is not found in the artifact cache, a request is issued (4) to get a list of repositories to search to find the artifact version.
- The Commander server replies (5) with a list of repositories and the order in which to search them.
- The step requests the artifact version from the first (6) repository in the list.
- If the artifact version is found in the repository, the artifact version is retrieved to the cache (7).
- If the artifact version is not found, the step requests the artifact version from the next repository in the search order (8).
- If the artifact version is found in that repository, the artifact version is retrieved to the cache (9).
- Otherwise, if the artifact version is not found in any repository, an error is displayed.



In this example, if the requested artifact version was found in the artifact cache (step 3), the resource would not have communicated with either of the repository servers. Also, any potential network latencies would have been avoided from retrieving the artifact version from the repository server.

If you have a cache shared by multiple resources and steps that run on those resources need to use the same artifact version, the first step requesting an artifact version retrieves it from the repository to the cache. Subsequent job steps using the artifact version will have immediate access to the artifact version in the cache, eliminating the need for each step to retrieve its own copy of an artifact version.

Cleaning up repositories and caches

Deleting an artifact version in the Commander server does not delete it in the repository backingstore, and it does not remove the artifact version from an artifact cache.

ectool and the ElectricCommander Perl module include a function for each of these tasks:

`cleanupRepository` and `cleanupArtifactCache`. Both of these API calls are intended to be run on the machine containing the `cache/backingstore` directory that needs to be cleaned up.

The cleanup algorithm is as follows:

1. Review the directory tree looking for directories three-levels deep.
2. For each such directory, interpret the three path components as `<groupId, artifactKey, version>`.
3. Issue a `findObjects` request to the Commander server to see if an artifact version with that specification exists.
4. Delete all directories identified by `<groupId, artifactKey, version>` and not recognized by the Commander server.

To operate correctly, using the `cleanupRepository` API **requires** read access to all artifact version objects stored in Commander and therefore must be run in an "admin" session. Electric Cloud also recommends running `cleanupArtifactCache` in an "admin" session—if not, artifact versions that are simply not readable by the user invoking the cleanup will be deleted from the cache.

In addition, both functions fail if the server does not recognize any of the chosen artifact versions. Thus, if the cleanup function is called on a directory that is not a backingstore directory or an artifact cache, it will not delete anything.

The recommended approach for removing stale artifact versions from a backingstore is to use a Commander maintenance procedure that periodically runs on each repository server (which presumably has an agent install). Similarly, use a procedure with a broadcast step to run on each agent to clean up its cache. Note that the job step session does not necessarily have read privileges on all artifact versions, so it is important to login as a privileged user, then perform cleanup with those credentials.

Authenticating Users for LDAP and Active Directory

ElectricCommander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository. **Local** users and groups are defined within ElectricCommander.

Use the following LDAP information and examples to fill-in your LDAP template. If you are using the Active Directory template, you may still use the LDAP information as reference—the templates are similar with only a slight difference in the Active Directory template.

Configuring LDAP

A number of options need to be customized for any LDAP configuration. The following sample configuration shows how we have our own LDAP configuration set up. After the sample, see a list of properties with a description.

Sample LDAP Configuration

```
<bean id="LdapDirectoryProvider"
    class="com.electriccloud.security.ldap.
    LdapDirectoryProviderImpl">
    <property name="providerName" value="LDAP"/>
    <property name="url" value="ldap://dir.
        electric-cloud.com/dc=electric-cloud,dc=com"/>
    <property name="managerDn" value="uid=JohnDoe,ou=People,=
        electric-cloud,dc=com"/>
    <property name="managerPassword" value="****"/>
    <property name="userBase" value="ou=People"/>
    <property name="userSearchFilter" value="uid={0}"/>
    <property name="userNameAttribute" value="uid"/>
    <property name="fullUserNameAttribute" value="gecos"/>
    <property name="emailAttribute" value="mail"/>
    <property name="groupBase" value="ou=Group"/>
    <property name="groupMemberFilter" value="(| (member={0})
        (memberUid={1}))"/>
    <property name="groupMemberAttributes" value="member,
        memberUid"/>
    <property name="groupSearchFilter" value="(| (
        groupOfNames) (objectClass=posixGroup))"/>
    <property name="groupNameAttribute" value="cn"/>
```

</bean>

The following properties configure LDAP mapping:

`emailAttribute` - (optional) The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.

`fullUserNameAttribute` - (optional) The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.

`managerDn` - (optional) The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access.

Note: This user does not need to be an admin user with modify privileges.

`managerPassword` - (optional) If the `managerDn` property is set, this password is used to authenticate the manager user.

`providerName` - This human readable name will be displayed in the user interface to identify users and groups that come from this provider.

`userBase` - This string is prepended to the `basedn` to construct the directory DN that contains user records.

`userNameAttribute` - The attribute in a user record that contains the user's account name.

`userSearchFilter` - This LDAP query is performed in the context of the user directory to search for a user by account name. The string "`r;{0}`" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.

`url` - The LDAP server URL is in the form `protocol://host:port/basedn`. Protocol is either `ldap` or `ldaps` for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for `ldap`, 636 for `ldaps`). The `basedn` is the path to the top level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a `dc=` and separated by commas.

Note: Spaces in the `basedn` must be URL encoded (`%20`).

In addition to user information, the LDAP server can be queried for group information. This query is optional because the local group mechanism can refer to LDAP users as well as *local* users. However, the following elements can be used to tell the server how to map groups in LDAP

`groupBase` - (optional) This string is prepended to the `basedn` to construct the directory DN that contains group records.

`groupMemberAttributes` - (optional) A comma separated attribute names list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.

`groupMemberFilter` - (optional) This LDAP query is performed in the groups directory context to identify groups that contain a specific user as a member. There are 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: "`r;{0}`" is replaced with the full user record DN, and "`r;{1}`" is replaced with the user's account name.

`groupNameAttribute` - (optional) The group record attribute that contains the name of the group.

`groupSearchFilter` - (optional) This LDAP query is performed in the context of the groups directory to enumerate group records.

Determining LDAP Mapping

A typical POSIX user record in LDAP looks similar the example below. To set up a mapping for this record, it is necessary to identify various record components. First, identify the path in the directory that contains user records. In this example, the build user has a distinguished name (dn) of `uid=build,ou=People,dc=mycompany,dc=com`. This name uniquely identifies the build user account and this path splits into three parts:

base dn: `dc=mycompany,dc=com`

user base: `ou=People`

user element: `uid=build`

The `baseDn` is the parent of the directory that contains users. This value should be combined with the protocol and server to form the URL. In this case, the URL is `ldaps://dir/dc=mycompany,dc=com`.

Next, the `userBase` is the portion of the path that identifies the directory containing all user account records. This value is used directly as the `userBase` configuration element.

The remaining portion identifies the user without the People directory: `uid=build`. The user name is replaced in this value with the string "`r;{0}`" to form the `userSearchFilter`: `uid={0}`. This query allows the server to search for a user's account name by looking for a record with a matching `uid` attribute.

The final mapping step is to identify user record attributes that hold the account name, full user name, and (optionally) the user's email address. In this example, the account name is `uid` (identified earlier), the full user name attribute is `gecos`, and there is no email attribute.

At this point, the server is able to authenticate a user, look up a user by name, and determine the user's full name. For many installations this is sufficient.

Sample LDAP User Record

```
# build, People, electric-cloud.com
dn: uid=jdoe, ou=People, dc=mycompany,dc=com
loginShell: /bin/bash
uidNumber: 508
gidNumber: 508
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
uid: jdoe
gecos: John Doe
cn: John
homeDirectory: /net/filer/homes/build
```

Also, you can configure the server to look for **LDAP groups** that refer to user accounts. A typical group record is shown below. Like a user account, an LDAP group has a distinguished name with a `baseDn`, a group base, and a group element. In this case, the `basedn` is still `dc=mycompany,dc=com`. The `groupBase` configuration element is `ou=Group`, and the group name is `cn=build_users`.

The server needs to identify records in the directory that correspond to groups—it does this by applying the `groupMemberFilter` to the records in the `groupBase` directory. In this case, group records are members of the `posixAccount` object class, so the filter can be set to `objectClass=posixGroup`. To display a group by its name, the server needs to know which attribute represents the group name. In this case, set the `groupNameAttribute` to `cn`.

Finally, the server needs a filter to determine which accounts belong to the group and the attribute name that represents a single group member. Group membership can be identified in one of two ways. Either the members are listed by account name, or by their LDAP distinguished name. In this case, POSIX group membership is determined by account name, so set the `groupMemberAttributes` property to `memberUid`, and set the `groupMemberFilter` to `memberUid={1}`.

Sample LDAP Group Record

```
# build_users, Group, mycompany.com
dn: cn=build_users,ou=Group,dc=mycompany,dc=com
objectClass: posixGroup
objectClass: top
gidNumber: 100
memberUid: jdoe
      memberUid: mary
cn: build_users
```

Sample Active Directory Configuration File

The following XML file defines parameters needed to connect to an Active Directory server and the query to use for looking up user information.

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:
      xmlns:tx
      xmlns:aop
      xsi:schemaLocation
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.0.xsd" default-lazy-
        init="true"

    <bean id="ADDirectoryProvider"
        class="com.electriccloud.security.ldap.ActiveDirectoryProviderImpl">
        <property name="providerName" value="ActiveDirectory" />
```

```
<!-- START OF CUSTOMIZATIONS -->

<!-- URL to AD server of the form: "r;ldap://host:port/base_dn" -->
<property name="url" value="ldap://server:389/dc=company,dc=com" />

<!-- Required credentials for an account that has read-only access to the server
-->

<property name="managerDn"
value="cn=myuser,cn=Users,dc=company,dc=com" />
<property name="managerPassword" value="mypw" />
<property name="userBase" value="cn=Users" />
<property name="userSearchFilter" value="(&(sAMAccountName={0})
(objectClass=user))" />
<property name="userNameAttribute" value="sAMAccountName" />
<property name="fullUserNameAttribute" value="name" />

<!-- Optional group configuration. This is only needed if you intend -->
<!-- to manage groups externally via AD. -->
<!-- <property name="groupBase" value="value=""/>-->
<!-- <property name="groupMemberFilter" value="member={0}"/> -->
<!-- <property name="groupMemberAttributes" value="member"/>-->
<!-- <property name="groupSearchFilter" value="(objectClass=group)"/>-->
<!-- <property name="groupNameAttribute" value="cn"/>-->

<!-- Max number of results to get back in one query. This must be -->
<!-- no greater than the AD server's setting, which is typically 1000. -->
<property name="pageSize" value="500" />

<!-- END OF CUSTOMIZATIONS -->

</bean>

</beans>
```


Credentials and User Impersonation

[Defining a credential](#)

[Why use credentials?](#)

[Credential access control](#)

[Using credentials for impersonation](#)

[Setting the impersonation credential](#)

[Accessing credentials from a step](#)

[Attaching a credential to a step](#)

[Passing credentials as parameters](#)

[Credential references](#)

[Best Practices for retrieving credentials](#)

What is a credential?

In ElectricCommander, a "credential" is an object that stores a user name and password for later use.

Two credential types are available, *stored* or *dynamic*:

- **stored** credentials - These credentials are given a name and stored in encrypted form in the database. Each Project has a list of stored credentials it owns. These credentials are managed from the Project Details page.
- **dynamic** credentials - These credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.

Defining a credential

After a credential is created, no one can view the password for the credential's account. This means one person can define a credential and enter the password, then other people can use the credential (and its account) without needing to know the password.

To create a new credential in the ElectricCommander UI:

1. Click the Projects tab.
2. Select a project (first column) to see the Project Details page.
3. Select the Credentials tab and then click the **Create Credential** link.
Click the **Help** link on the New Credential page if you need more information to fill-in the fields.

Why use credentials?

Use ElectricCommander credentials for:

- User impersonation
- Saving passwords for use inside steps

When a step needs to run as a particular user, ElectricCommander can retrieve the username and password from a stored credential. The credential is passed to the agent over an encrypted channel so the agent can authenticate itself to the operating system and set up a security context where the step runs with the user permissions in the credential.

In some cases, a step needs to supply credentials directly to an application.

- The credential may be a fixed application credential that needs to be presented every time the step runs. In this case, a stored credential provides a place to put this information without needing to embed the password in a script.
- Other times, the password is not known in advance. In this case, a dynamic credential created when a job is launched can prompt a user for information and then pass the credential to the step in a secure manner.

Credential access control

Because ElectricCommander stores passwords in a recoverable manner, it is important that credentials are accessible only under carefully controlled circumstances. To protect credentials, ElectricCommander uses two mechanisms access control lists (ACL) and *attached* credentials.

To use a credential for any purpose, a user who attempts to reference the credential must have execute permission on the credential object—this allows you to explicitly control which users are allowed to use a credential anywhere in the system.

In addition to the user-based access control check, ElectricCommander also requires a credential be explicitly *attached* to the object that is going to use it. After an object has a credential attached, object modifications are restricted to users who have both execute permission on the credential and modify permission on that object. This safeguard helps prevent credentials from accidentally being used by unauthorized users.

Two ways to attach a credential to an object:

- Set the impersonation credential for a Step, Procedure, Project, or Schedule. Impersonation credentials are inherited, so attaching a credential to a container object like a Procedure or a Project implicitly attaches it to every object inside the container, for example, all steps in a procedure.
- Explicitly attach the credential to a step or schedule using the UI or the `attachCredential` API. To access credentials from a step, the credential must be attached directly to the step, or the credential must be passed as a parameter to the containing procedure and have the parameter attached instead.

Using credentials for impersonation

This information addresses the question, "When a job step is running on a resource, under which account is it running and how can I control that?" By default, a job step runs under the account used by the ElectricCommander agent that runs the step, which was chosen when the agent was installed. This approach works well in environments where it makes sense to run all jobs under a single user such as a "build" user. You install all agents to run as the desired user, then you do not have to worry about anything else— every step of every job runs as that user.

However, in other environments you may prefer to run different jobs, or even different job steps, under different accounts

For example:

- If independent groups are sharing an ElectricCommander system, you may want each group to use a different account for its jobs.
- Or, you may need to run jobs as particular users to access ClearCase views for those users.
- Or, there may be certain steps that require special privileges.
For example, as part of your builds you may need to run a step that generates a certificate using privileged corporate information; that step must run under a special high-privileged account, but you want the remainder of the steps in the build to use a less-privileged account.

ElectricCommander allows you to select accounts on a per-job or per-job-step basis—this mechanism is called *impersonation* (the ElectricCommander agent impersonates a particular user for the duration of a job step), and this *impersonation* is implemented using credentials. The use of credentials creates special security challenges.

Note: It is important to ensure privileged accounts can be used only for the purposes you intend and cannot be "hijacked" for other purposes. ElectricCommander's access control mechanism contains special facilities to ensure proper credential use.

Setting the impersonation credential

You can attach impersonation credentials to procedure steps, procedures, and projects before executing a job step. ElectricCommander searches for a credential—first in the procedure step, then in its procedure, then in its project, and uses the first credential found.

If the step is in a nested procedure, and no credential is found on the step, its procedure, or its project, ElectricCommander checks the calling step (and its procedure and project), then its caller, and so on until it has worked up through the topmost procedure in the job. If no credential is found, the job step runs under the ElectricCommander agent account on its resource.

This approach makes it easy to manage your account usage. For example, if you want all jobs in one project to use a particular account, define a credential in that project for the account, then attach the credential to the project. If you want all jobs in a project to use a particular account, except for one step that should use a different account, create two credentials in the project. Attach the first credential to the project and attach the second credential to the particular step.

You can attach a credential to a schedule also. If you do, the credential in the schedule is used for steps where no other credential is available. The schedule's credential receives the lowest priority. Finally, you can specify a credential when you launch a job manually.

You can do this in one of two ways:

- you can provide the name of an existing credential
- you can enter an account and password

If you choose the second approach, the account and password are stored only in ElectricCommander for the duration of the job. In either case, the credential you specified when you launched the job is used as a last resort for steps only with no other credential.

To attach a new impersonation credential:

To attach an impersonation credential for a procedure, step, or schedule, begin by clicking the Projects tab, then select a project to go to the Project Details page. Continue as follows:

- For a **procedure** - click the **Create Procedure** link to go to the New Procedure page. On this page, select the Impersonation Credential drop-down box to select a credential to use for impersonation.
- For a **step** - select a procedure to go to the Procedure Details page, then select a step name to go to the Edit Step page. On the Edit Step page, scroll down to the Impersonation section.
- For a **schedule** - select the Schedules subtab. Select a schedule to go to the Edit Schedule page, scroll down to Impersonation Credential section.

For more information, each of these web pages contains a **Help** link in the upper-right corner.

Accessing credentials from a step

The ElectricCommander server supports an operation called `getFullCredential` that can return a password when called from inside a step. The ectool interface for this operation is:

```
getFullCredential <credential name>
    [--value <password|user>]
```

Without the `--value` option, the response is the same as that returned by `getCredential`, with the addition of a "password" element.

With `--value`, the simple text password or username value is returned on *stdout* so it can be used directly without XML parsing.

The `getFullCredential` API is allowed only inside a running step, and is allowed to use credentials that were explicitly attached to the step only.

Attaching a credential to a step

A credential must be explicitly attached to the object using it so the server can perform an access control (ACL) check at definition time and limit the visibility of the password. To support accessing credentials other than the one being used for impersonation, steps, procedures, and schedules contain a list of credential names.

In the ElectricCommander UI, to attach a credential [other than an impersonation credential], you must be on an "Edit" page. For example, to create a credential for a step, a step must already exist. Select a step [that needs a credential] to go the Edit Step page, scroll to the lower section of the page and click the **Attach Credential** link.

If you are an ectool command-line user, you can use the `attachCredential` API command. For more information, you can access the *ectool* help topics within the online help system.

Passing credentials as parameters

Sometimes you may not know which credential you need to use when you define a step in a procedure. In this case, you can leave the credential choice up to the caller of the procedure.

The following example describes the procedure for passing credentials as parameters:

1. Create a project named InnerProj.
2. Within that project create a procedure named InnerProc with a parameter 'cred' of the type credential.

3. Within that procedure create a step named InnerStep with:
 - Command: `ectool getFullCredential cred`
 - Attached Parameter Credential: `cred`
4. Create a project named OuterProj with two credentials: `cred1` and `cred2`.
5. Within that project create a procedure named OuterProc with two parameters: '`cred1`' and '`cred2`', of the type credential.
6. Within that procedure create two subprocedure steps that both call InnerStep from the InnerProj project:
 - OuterStep1 with Attached Parameter Credential `cred1`, and `cred1` in the Parameters section
 - OuterStep2 with Attached Parameter Credential `cred2`, and `cred2` in the Parameters section

If you are an ectool command-line user, you can use the `createFormalParameter` and `attachParameter` API commands. For more information, see the *ectool* help topics within the online help system.

Credential references

For schedules, the value for any actual parameter passed to a credential-type formal parameter is interpreted as a reference to a previously attached credential on the schedule.

For procedure steps, the value for any actual parameter passed to a credential type formal parameter is interpreted as a reference to a previously attached credential or credential parameter on the calling step.

When a job step is created, all credential references are resolved and the content is copied into transient credentials on the job step. Credential parameter references are resolved by looking for a credential in the calling job step's transient credential map—looking for a credential with the name specified as the actual parameter value.

Credential references in the API can be specified as a relative or an absolute credential path.

- A *relative* path is similar to "rootCred" and is interpreted as referring to a credential in the current project.
- An *absolute* path includes a project name like `"/projects/MyProject/credentials/rootCred"` and can refer to a credential in any project.

Anywhere a credential reference is accepted (for example, setting an impersonation credential or attaching a credential to a step), either form can be used.

Best Practices for retrieving credentials

ElectricCommander's impersonation mechanism provides powerful mechanisms for using different accounts in job steps, and it can be used to handle a variety of challenges. However, if misused, impersonation can open up dangerous security loopholes. This section discusses these risks and how to avoid them.

Use of credentials and impersonation tend to fall into two classes. In the first class, your goal is to open up access to accounts, you want to make some accounts generally available, and you trust large groups of users (such as everyone who can access a particular Project) to use the accounts appropriately. This usage type is simplest and relatively safe. First, define an ElectricCommander group containing all trusted users. Next, set permissions on the Project so only trusted users can access the Project. Finally, define credentials for accounts in the Project and grant execute permission to everyone in the group.

After completing this task, everyone in the group can use the accounts for arbitrary purposes. No one outside the group will be able to access the Project or the credentials.

The second use-case for credentials is when you want to provide tightly-controlled access to a highly privileged functions such as generating a certificate or accessing sensitive information. This kind of usage requires careful thought to avoid allowing unintended access to the credential. The best way to implement a solution for this scenario is to restrict access to the credential and use it on individual steps only after careful analysis of issues such as the following:

Does the step's command use properties?

If so, someone could potentially change the property value to change the step behavior.

For example, suppose the step's command is defined like this:

```
echo $[/myProject/x]
```

Anyone with write access to property `x` can hijack the step by placing the value `"nothing; myCommand"` in the property.

After property substitution, the final command will be:

```
echo nothing; myCommand
```

which will cause `"myCommand"` to execute in the step.

Either restrict access to the property sheet or avoid substitutions.

Does the step execute code or commands that can be modified?

In most jobs that execute build and test sequences, the job extracts various files from a source code control system and some of those files contain code executed during the build. In this case, anyone with access to the source code control system can affect the executed code.

For example, suppose a step runs a Make or Ant command using a configuration file extracted from the source code control system. Anyone with access to the source code control system can modify the configuration file to introduce new commands executed during the Make or Ant step. In this situation, it is virtually impossible to control what happens during the step, so you should never attach a credential for a privileged account to such a step.

The higher you attach a credential, the greater the risk of uncontrolled access to the credential's account.

For example, if you attach a credential to a project, there is a good chance anyone with write access to the project, or even the ability to execute the project's procedures, can hijack the credential using one of the loopholes described above. If you attach a credential to a step that invokes a subprocedure, you effectively give anyone with write access to the subprocedure complete access to the credential's account. If you care about access to an account, you should use its credential in the narrowest possible fashion, such as attaching it to a single job step.

Avoid passing passwords on the command line

When using credentials, avoid passing the password on the command line because [on most platforms] those passwords will be visible to all users logged into the system. The secure way to pass a credential to an application is to use a secured file in the file system or to pipe the value into the application via `stdin`.

Customizing the Commander UI

You can customize the Commander UI to get a more intuitive, task-specific user interface.

[Customizing parameters](#)

[Customizing tab layouts](#)

[Home page configurations](#)

Customizing parameters

Two types of objects contain formal parameters – procedures and state definitions. For these objects, you can customize the way their formal parameters are presented. You can reorder the parameters and set the labels, form element types, default values, tooltip help text, and whether or not the parameter is required. When defined, this customization shows up whenever the parameters are displayed—running a procedure, creating a step to call a procedure, creating a transition definition to a specific target state, and so on.

How do you customize parameters?

Create a property named `ec_parameterForm` on the object containing formal parameters (the procedure or state definition). The value of this property is an XML-style specification of form elements that map to the parameters. This property must be in sync with the formal parameters, which means all formal parameters must have a corresponding XML element, with no "extra" XML elements.

Custom parameter form contents

The value of the property in XML: Under a top-level element called `<editor>`, you need an element called `<formElement>` for each parameter. Under a `<formElement>`, you can specify the following tags:

- `property*` - the name of the formal parameter
- `type*` - `entry|textarea|select|radio|checkbox|project|savedSearch|credential`
- `label*` - displayed next to the element
- `value` - the initial value of the element (does not apply to credentials, which must be specified at run time)
- `required` - if true, the user must enter a value for the element
- `documentation` - text displayed in the tooltip when the form element is "moused" over
- `numRows` - valid for `textarea` elements; represents the height of the element
- `option` - a single option for `select|radio` elements; at least one is required
 - `name*` - the text displayed in the option
 - `value` - the value of the option
- `checkedValue` - valid for `checkbox` elements; the value of the element when it is checked

- uncheckedValue - valid for `checkbox` elements; the value of the element when it is unchecked
- initiallyChecked - valid for `checkbox` elements; whether or not the element is "checked" by default

Note: An asterik (*) in the list above indicates a *required* tag.

Example

The following example is a parameter form that rearranges parameters and sets the labels, descriptions, default values, and form elements for each parameter.

```
<editor>
  <formElement>
    <label>One:</label>
    <property>one</property>
    <documentation>The first parameter.</documentation>
    <type>entry</type>
    <value>Test value</value>
  </formElement>
  <formElement>
    <label>Two:</label>
    <property>two</property>
    <documentation>The second parameter.</documentation>
    <type>textarea</type>
    <required>1</required>
  </formElement>
  <formElement>
    <label>Three:</label>
    <property>three</property>
    <documentation>The third parameter.</documentation>
    <type>select</type>
    <option>
      <name>ABC</name>
      <value>abc</value>
    </option>
    <option>
      <name>XYZ</name>
      <value>xyz</value>
    </option>
    <value>xyz</value>
  </formElement>
  <formElement>
    <label>Four:</label>
    <property>four</property>
    <documentation>The fourth parameter.</documentation>
    <type>radio</type>
    <option>
      <name>First Option</name>
      <value>123</value>
    </option>
    <option>
      <name>Second Option</name>
      <value>456</value>
    </option>
    <value>123</value>
  </formElement>
</formElement>
```



```

    <label>Five:</label>
    <property>five</property>
    <documentation>The fifth parameter.</documentation>
    <type>checkbox</type>
    <checkedValue>true</checkedValue>
    <uncheckedValue>false</uncheckedValue>
    <initiallyChecked>1</initiallyChecked>
    <value>true</value>
  </formElement>
  <formElement>
    <label>Six:</label>
    <property>six</property>
    <documentation>The sixth parameter.</documentation>
    <type>project</type>
    <value>myProject</value>
  </formElement>
  <formElement>
    <label>Seven:</label>
    <property>seven</property>
    <documentation>The seventh parameter.</documentation>
    <type>savedSearch</type>
    <value>/projects/myProject/ec_savedSearches/mySavedSearch</value>
  </formElement>
  <formElement>
    <label>Eight:</label>
    <property>eight</property>
    <documentation>The eighth parameter.</documentation>
    <type>credential</type>
  </formElement>
</editor>

```

Customizing the tab layout

Overview

A "view" defines the layout of tabs in the Commander web UI. One or more tab views may be defined at the server, group, and user level. The default view the user sees when they first log in can be set at the server, group, and user level. The default defined on the user takes precedence over its groups and the groups take precedence over the server. Electric Cloud provides a system default view that will always show up in the user's list. Views can inherit from each other and add/modify/remove/reposition tabs and subtabs as needed.

If you define a custom view, you have to manually add tabs such as Continuous Integration so that they will appear in your custom view.

View definition syntax

A view is an XML property. The following is a list of elements that can be defined in a view:

- tab - a top level tab
 - label - the text to display for the tab
 - url - the target URL of the tab

- accesskey - the keyboard shortcut to access this tab (a single letter)
- position - the position of this tab relative to the other tabs (first tab is 1, second tab is 2, and so on)
- show - when inheriting from another base view, if 0, the tab is not visible, if 1 it is visible (default 1)
- tab - a subtab displayed within this tab
 - label - the text to display for the subtab
 - url - the target URL of the subtab (relative to Commander base)
 - position - the position of this subtab relative to the other subtabs (integer greater than 1)
 - show - when inheriting from another base view, if "0", the tab is not visible, if "1" it is visible (default 1)
- base - the name of a view from which to inherit tab and subtab definitions

The following is a basic definition of a two-tab view where one of the tabs has three subtabs.

```
<?xml version="1.0" encoding="utf-8"?>

<view>
  <tab>
    <label>Home</label>
    <url>home.php</url>
    <accesskey>h</accesskey>
  </tab>
  <tab>
    <label>Administration</label>
    <url>workspaces.php</url>
    <accesskey>a</accesskey>
    <tab>
      <label>Workspaces</label>
      <url>workspaces.php</url>
    </tab>
    <tab>
      <label>Directory Providers</label>
      <url>directoryProviders.php</url>
    </tab>
    <tab>
      <label>Licenses</label>
      <url>licenses.php</url>
    </tab>
  </tab>
</view>
```

The following is a view that inherits from the above view and:

1. Changes the URL of the Home tab
2. Adds a new tab called Projects and places it at the beginning of the list
3. Changes the accesskey of the Administration tab
4. Moves the Workspaces subtab to the end of the list
5. Hides the Licenses subtab

```
<?xml version="1.0" encoding="utf-8"?>

<view>

  <base>firstView</base>

  <tab>

    <label>Home</label>

    <url>customizedHome.php</url>

  </tab>

  <tab>

    <label>Projects</label>

    <url>projects.php</url>

    <accesskey>p</accesskey>

    <position>1</position>

  </tab>

  <tab>

    <label>Administration</label>

    <accesskey>z</accesskey>

    <tab>

      <label>Workspaces</label>

      <position>3</position>

    </tab>

    <tab>

      <label>Licenses</label>

      <show>0</show>

    </tab>

  </tab>

</view>
```

Storing views

Tab views are stored in the server, group, and user property sheets. The view definitions are stored in a property sheet called `ec_ui/availableViews`. The name of the view is set to the property's description if defined, otherwise it is set to the property's name. The value of the view property is the XML definition document described above.

Default views

The property `ec_ui/defaultView`, if set, determines the default view for users inheriting from that object.

- If this property is set on the server, it is the default for all users.
- If this property is set on a group, all users belonging to that group will see that view, overriding the server's default if it is set also.
- Finally, a user can explicitly set their view by defining the property on their own property sheet. If the user belongs to multiple groups that define defaults, then Commander chooses the first group alphabetically.

The user can set their view by clicking on their name in the navigation bar, then clicking on the **Edit Settings** link. The list of views shown here comprise all views defined in the `ec_ui/availableViews` property sheets for the server, groups to which the user belongs, and the user itself.

If views have the same name at different levels, then the user overrides the group which overrides the server. If a user selects a view and chooses to Save, then the `ec_ui/defaultView` property on their sheet is set accordingly.

Two special values are always available in this list: "EC Default" and "Inherit". The former is the default tab layout defined in the Commander web UI. The latter means to use the first view in the list. By default, all users inherit their tab view.

For more information on user settings, see the [Edit User Settings](#) help topic.

Developing and troubleshooting

If you create an invalid view definition on a group or server property sheet, you will break the UI for all users who inherit that view. The best practice for developing tab views is to use a "test user". Store the view definition in the `ec_ui/availableViews` property sheet for that user, and set the `ec_ui/defaultView` for that user to the name of the new view. When the view is working properly and ready for other users, you can migrate the new view to a group or to the server.

If you continue to have difficulty and tab definitions are not working correctly and can no longer get to the user settings page, revert to the Commander default by running the following command:

```
ectool setProperty /users/$USERNAME/ec_ui/defaultView ec_default
```

When you log out and then log back in, you will have the default view again.

Note that changes to views are not visible until the next time a user logs in.

Home page configuration

The Commander Home page is your configurable Dashboard. This page allows you to manage shared configurations and you can customize this page for your work preferences also. Refer to [The Home page](#) Help topic for details on creating Job Configurations, Shortcuts, Jobs Quick View, and Reports.

Shared Home page configurations can be set globally or for specific groups of users only. The general model for creating shared configurations is:

1. Set up your personal configuration the way you want it to appear to others.
To get to the configuration page, select the Administration tab > Plugins > and click the **Configure** link for EC-Homepage.
2. Click **Save** to make the configuration available to a specific group or globally to everyone on the server.

If you published the configuration to a specific group, no further action is needed. If you published the configuration globally, continue to the next step.

3. Copy and paste the Tab XML string into a specific user's or group's view.

This adds the public configuration to that user's or group's own home page and no further action is required for the user or group. If this step is not completed, additional action is required as described in [Installing the Home page](#).

You can alter an existing shared configuration by loading from a previously saved location, make changes, then save the changes back to the original or an alternate location.

User settings

Use the Backup Settings action to save and restore your personal Home page configuration.

Select **Create** to back up and temporarily set aside your personal settings while you create or update shared settings.

Select **Restore** to retrieve your personal settings after updating shared settings.

Location

Select Global to share the new Home page with everyone on the server, or select Group to share with specific users only.

Installing the Home page

If a globally available Tab XML string (from the Configure EC-Homepage page) has **not** been pasted to the user's or group's view, it must be added to an existing view definition to replace the standard Home page with the one provided by this plugin. To do this, click **Load**. This replaces the current home page with the one that's publicly available.

Reconfigure Contact Support link

In the event you want to redirect the Electric Cloud Support URL (available from the Help link on each Commander web page) to your own support center, you can create a file to add to the Commander installed directory at `<Commander Install Dir>/apache/htdocs/commander`. Create a file named `config_user.php` with the following contents:

```
<?php
$config["contactSupportUrl"] = "your URL";
?>
```

Notes:

1. You must supply the value (URL) for your customer support site, replacing "your URL" in the example above. If no value is supplied, a blank window is displayed.
2. You must restart the Web Server for your change to take effect.

Defect Tracking

The Defect Tracking plugin enables linking existing defects to an ElectricCommander job. "Existing defects" are those defects previously created in the defect tracking system and already associated with the Commander job in some way.

For example, a defect is associated with a Commander job if the fix for the defect is part of the source code snapshot being built and tested in the job.

Commander pre-installs numerous defect tracking plugins including Bugzilla, ClearQuest, Fortress, JIRA, Quality Center, Ration Team Concert, Rally, Team Foundation Server, TeamForge, TestTrack, and more, using plugin integrations.

The following examples and instructions are for a JIRA integration. The steps for other defect tracking systems are similar to those for JIRA.

Scenario example

A developer fixes a defect in the "ABC" project and checks in the fix to the source control system, along with a comment that includes the fixed defect ID, for example:

```
ABC-123: fixed EFG bug
```

When the next Commander job is triggered for the ABC project, Commander checks out a source code snapshot from the source control system and queries the source control system for a log containing check-in details. This log, which should contain the above comment, will be stored in a property on the job.

The JIRA plugin will then do the following:

- Parse the property containing the source control system log to identify defect IDs.
- Query the configured JIRA server with identified defect IDs, including "ABC-123".
- Construct a descriptive URL to point to the JIRA defect on the JIRA server.
- Include the URL in the JIRA Report.
- Link to this JIRA Report from the Job Details page.

Example: Enabling the JIRA integration in your procedure

To ensure Commander links existing defects to a job, create a step to link the defects.

Go to Projects > select a Project > select a Procedure. To create a New Step, select the **Plugin** link.

- In the Choose Step panel, select Defect Tracking from the left pane, then select the defect tracking system you configured.
- The right-pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the defect tracking integration you configured and the step you chose.

On the New Step page, fill-in the following fields:

General section:

Name - Supply a unique name for your subprocedure step - any name of your choice.

Description - (optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a>` `` `
` `<div>` `<dl>` `` `<i>` `` `` `<p>` `<pre>` `` `<style>` `<table>` `<tc>` `<td>` `<th>` `<tr>` ``

Subprocedure section:

Resource - (optional) You do not need to change the resource you previously chose to run the procedure.

Parameters section:

config - (required) This is the name of the Defect Tracking configuration you created on the New Defect Tracking Configuration page.

Note: If you did not previously create a Defect Tracking configuration, you must do that now before you proceed. Go to Administration > Defect Tracking > click the **Create Configuration** link to see the New Defect Tracking Configuration page. Access the **Help** link on that page for more information.

prefix - (optional) This is the key used by JIRA as the prefix for defects within a project. If the "prefix" field is blank, a regular expression is used to try and match defect IDs.

propertyToParse - (optional) This is the property or property sheet to search for defect IDs. If the "propertyToParse" field is blank, the default property `/myJob/ecscm_changeLogs` is used.

Advanced section:

You do not need to supply any information in these fields.

The following screen is an example of the New Step page where you will create your Defect Tracking subprocedure step.

New Step



General

Name:

Description:

Subprocedure

Subprocedure: EC-DefectTracking-JIRA : LinkDefects [Change](#)Resource: [Browse](#)

Parameters

config: Requiredprefix: propertyToParse:

Advanced

Run Condition:

Error Handling: Procedure continues, but overall status will be error Time Limit: minutes Run in Parallel: ☐Always Run Step: ☐Retain Exclusive: NoneRelease Exclusive: NoneWorkspace: [Browse](#)

Impersonation

Credential Project: ☒ Current ☐ [Browse](#)Credential Name: [Browse](#)

Defect Tracking

The Defect Tracking plugin enables linking existing defects to an ElectricCommander job. "Existing defects" are those defects previously created in the defect tracking system and already associated with the Commander job in some way.

For example, a defect is associated with a Commander job if the fix for the defect is part of the source code snapshot being built and tested in the job.

Commander pre-installs numerous defect tracking plugins including Bugzilla, ClearQuest, Fortress, JIRA, Quality Center, Ration Team Concert, Rally, Team Foundation Server, TeamForge, TestTrack, and more, using plugin integrations.

The following examples and instructions are for a JIRA integration. The steps for other defect tracking systems are similar to those for JIRA.

Scenario example

A developer fixes a defect in the "ABC" project and checks in the fix to the source control system, along with a comment that includes the fixed defect ID, for example:

```
ABC-123: fixed EFG bug
```

When the next Commander job is triggered for the ABC project, Commander checks out a source code snapshot from the source control system and queries the source control system for a log containing check-in details. This log, which should contain the above comment, will be stored in a property on the job.

The JIRA plugin will then do the following:

- Parse the property containing the source control system log to identify defect IDs.
- Query the configured JIRA server with identified defect IDs, including "ABC-123".
- Construct a descriptive URL to point to the JIRA defect on the JIRA server.
- Include the URL in the JIRA Report.
- Link to this JIRA Report from the Job Details page.

Example: Enabling the JIRA integration in your procedure

To ensure Commander links existing defects to a job, create a step to link the defects.

Go to Projects > select a Project > select a Procedure. To create a New Step, select the **Plugin** link.

- In the Choose Step panel, select Defect Tracking from the left pane, then select the defect tracking system you configured.
- The right-pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the defect tracking integration you configured and the step you chose.

On the New Step page, fill-in the following fields:

General section:

Name - Supply a unique name for your subprocedure step - any name of your choice.

Description - (optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a>` `` `
` `<div>` `<dl>` `` `<i>` `` `` `<p>` `<pre>` `` `<style>` `<table>` `<tc>` `<td>` `<th>` `<tr>` ``

Subprocedure section:

Resource - (optional) You do not need to change the resource you previously chose to run the procedure.

Parameters section:

config - (required) This is the name of the Defect Tracking configuration you created on the New Defect Tracking Configuration page.

Note: If you did not previously create a Defect Tracking configuration, you must do that now before you proceed. Go to Administration > Defect Tracking > click the **Create Configuration** link to see the New Defect Tracking Configuration page. Access the **Help** link on that page for more information.

prefix - (optional) This is the key used by JIRA as the prefix for defects within a project. If the "prefix" field is blank, a regular expression is used to try and match defect IDs.


propertyToParse - (optional) This is the property or property sheet to search for defect IDs. If the "propertyToParse" field is blank, the default property `/myJob/ecscm_changeLogs` is used.

Advanced section:

You do not need to supply any information in these fields.

The following screen is an example of the New Step page where you will create your Defect Tracking subprocedure step.

New Step



General

Name:

Description:

Subprocedure

Subprocedure:

EC-DefectTracking-JIRA : LinkDefects

Change

Resource:

Browse

Parameters

config:

default

Required

prefix:

propertyToParse:

Advanced

Run Condition:

Error Handling:

Procedure continues, but overall status will be error

Time Limit:

minutes

Run in Parallel:

☐

Always Run Step:

☐

Retain Exclusive:

None

Release Exclusive:

None

Workspace:

Browse

Impersonation

Credential Project:

☒ Current ☐

Browse

Credential Name:

Browse

OK

Cancel

ElectricSentry

ElectricSentry is the Commander **engine** for continuous integration, integrating with Source Control Management (SCM) systems. ElectricSentry is installed automatically with Commander and is contained in a Commander plugin named ECSCM. When ElectricSentry detects a new or modified source file check-in, it launches a user-defined procedure to build and test the latest version of source files. Using a few simple properties, the entire process is controlled by the user who determines when ElectricSentry is active and which projects and branches are monitored and built.

While you can interact with ElectricSentry directly, you may want to use the Continuous Integration Dashboard, *the front-end user interface* for ElectricSentry, which provides a visual display of your builds, and automates the configuration process for your branch, project, or procedure so you can start using continuous integration schedules quickly.

This help topic describes using ElectricSentry directly, and NOT the Continuous Integration Dashboard, which has its own help topic for dashboard use. To get started with the "dashboard", select the Home > Continuous Integration tabs.

How ElectricSentry works

ElectricSentry finds procedures the user wants to monitor by scanning for a special type of Commander schedule. From the schedule, ElectricSentry retrieves the name of the procedure to run, along with values to use as actual parameters when the procedure runs.

ElectricSentry can run at a regular interval you determine. When it runs, ElectricSentry scans all Commander projects, looking for schedules configured for use with ElectricSentry. When it finds one, ElectricSentry checks to see if there is already a job running based on that schedule and will not start a second job for the same schedule until the previous job completes.

Next, ElectricSentry queries the SCM system to see if new check-ins have occurred since the last attempted build. If the most recent check-in is different from the last attempted build, ElectricSentry then checks a "quiet period" by comparing the current time to the check-in time. If the user-configured time interval has elapsed since the most recent check-in, ElectricSentry runs the procedure to start a new job.

Finally, ElectricSentry deletes previous runs of itself, so only the most recent run appears in the jobs list. If using the Continuous Integration Dashboard, this is slightly different—you will see the previous 10 builds that ran.

Configuring ElectricSentry

ElectricSentry does not require any specific configuration, but it does allow fine tuning for some key settings. To change any setting, you must be logged into Commander and then navigate to the Electric Cloud project.

Quiet time

When setting up a continuous integration system, it is common to require an inactivity period before starting a build. This time period allows developers to make multiple, coordinated check-ins to ensure a build does not start with only some of the changes—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.

With ElectricSentry, the inactivity time period can be configured globally for all projects or individually for a single project. The global setup is stored in the ECSCM-SentryMonitor schedule in the Electric Cloud project, using a property named `ElectricSentrySettings/QuietTimeMinutes`. You can set this property to the number of minutes you prefer for the quiet time prior to starting a build. If you remove this property, ElectricSentry will use a 5-minute default time.

Resource

The ECSCM plugin contains a procedure named `ElectricSentry`. This procedure has a parameter named `sentryResource` that determines where `ElectricSentry` will run. The parameter defaults to a resource named `local`, which is created by the `ElectricCommander` installer. The local resource is configured to run on the machine named `localhost`, which refers to the machine where the Commander server is running.

You may need to change the resource where `ElectricSentry` runs, if for example, the server machine is not configured to run your SCM tools. You can change this configuration by modifying the ECSCM-SentryMonitor schedule to specify a different resource in the `sentryResource` parameter.

To do this:

1. Go to the Projects page.
2. Select the Electric Cloud project name to go to the Project Details page.
3. Click the Schedules subtab.
4. Select the ECSCM-SentryMonitor schedule.
5. Change the `sentryResource` field to the name of the resource (or the resource pool name) where you want `ElectricSentry` to run.
6. Click **OK** to save your new information.

Polling frequency

`ElectricSentry` is set to look for new check-ins every five minutes. If you want `ElectricSentry` to check more or less often, open the ECSCM-SentryMonitor schedule in the Electric Cloud project. Change the number of minutes to speed up or slow down the `ElectricSentry` polling frequency entry.

Time-of-day and day-of-week

The initial setting allows `ElectricSentry` to run between 7 am and 11 pm every day of the week, but you can change this time period to monitor check-ins at different hours. The time period should cover the most likely hours developers would be checking-in changes, then frees the resources (after the last job started by `ElectricSentry` finishes) to run overnight builds. You can modify the ECSCM-SentryMonitor schedule to change the hours or days `ElectricSentry` monitors check-ins.

Configuring a build for continuous integration

You can control which projects and branches `ElectricSentry` monitors for continuous integration by creating a schedule that runs a procedure and by adding `ElectricSentry` settings. These tasks are made easier using the Continuous Integration Dashboard.

Source Control Management (SCM) configurations

ElectricCommander bundles and supports a number of source control types. After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page—to see this web page, select the Administration > Source Control tabs.

Click the **Create Configuration** link to configure your source control system.

Note: Electric Cloud has tested numerous SCM versions and where possible, all SCM integrations were created in a generic manner to avoid SCM release-specific differences. In most cases, Electric Cloud supports any version of Perforce, ClearCase, Subversion, AccuRev, CM Synergy, Borland's StarTeam, and many others.

Select the Administration > Plugins tabs to go to the Plugin Manager page to the current list of available SCM plugins. If you do not see the SCM you need, it may be available in the Plugin Catalog—select the View Catalog tab.

Create a procedure

Navigate to your project and create a procedure that checks out sources from your SCM system and then runs your build process. Electric Cloud recommends having the procedure take a "branch name" parameter, then use this single procedure to build multiple branches of the same project.

Create a schedule

In your project, create a schedule that calls the procedure and specifies parameters to call the procedure. On the Project Details web page, select the Schedules tab and click the **CI Configuration** link to go to the New CI Configuration page. Enter a name for the new schedule and choose the SCM Configuration you want to use. For more information on this page, click the **Help** link in the upper-right corner of the screen. If you set up your procedure with a parameter for a branch, create one schedule for each branch you want to monitor.

Optional - running ElectricSentry on multiple resources

ElectricSentry normally runs as a single procedure executing queries against all SCM systems configured by ElectricSentry "trigger schedules." Because all queries are executed in a single step, these queries are executed by the same Commander resource. This process works very well for many companies, but for some large enterprise companies this process is not adequate.

You can benefit from using ElectricSentry on multiple resources if any of the following items apply to your organization:

- Connectivity - No one single resource has access to all SCM systems in your organization.
- Security/Authentication - Specific resources may contain Authentication tickets required for access to certain parts of your SCM depot(s), and no single resource contains all of these tickets.
- Scalability - You may need too many trigger schedules to monitor in a single execution interval, which can cause unnecessarily long execution times for ElectricSentry jobs.
- Scheduling - ElectricSentry is most useful during the work day when developers are adding new code check-ins. At night, it is common to redeploy resources to do nightly builds. If ElectricSentry runs on a central resource that serves development teams in multiple time zones, there is no way to schedule independent Sentry operation for different time zones.

Overview

The solution to all of these issues [above] is to run multiple ElectricSentry instances concurrently.

- Each Sentry instance can choose its own resource and its own operation schedule.
- Each instance can be restricted to monitoring schedules for one or more Commander projects.
- Also, you can create a global instance to monitor all projects not monitored by a restricted instance.

Each ElectricSentry instance is implemented by re-using the ElectricSentry procedure in the ECSCM plugin. This procedure has two parameters, `sentryResource` and `projectList`, that allow you to specify the configuration for different instances.

Each instance will have its own copy of the SentryMonitor schedule—one for each resource where you want ElectricSentry to run.

The default configuration has no projects specified in the "projectList" parameter—this is called the *global* instance. The global instance monitors all projects not explicitly monitored by any other instance. If a global instance is your only instance, it will monitor all projects.

Configuring ElectricSentry to use multiple resources

When ElectricCommander is installed, it creates a project called Electric Cloud that contains ElectricSentry's Sentry Monitor schedules. By default, all steps will run on the resource named local and all projects will be monitored. To use multiple resources, you need to create new schedules in the Electric Cloud project, creating new ElectricSentry instances.

1. Navigate to the Electric Cloud project and select it.
2. Select the Schedules tab.
3. Disable the schedule named ECSCM-SentryMonitor.
4. Click the **Copy** link next to the ECSCM-SentryMonitor schedule - **repeat** for each new ElectricSentry instance you want to create.
5. Re-enable the schedule named ECSCM-SentryMonitor.
6. Navigate to one of the new schedules you created and select it to edit it - change the name from ECSCM-SentryMonitor copy n to something more meaningful, for example, "ECSCM-SentryMonitor for ABC team".
7. In the Parameters section:
projectList box - enter the project names you need to monitor - one project name per line.
sentryResource box - enter the resource or pool name where you want this ElectricSentry instance to run.
8. In the Frequency section:
Adjust the "time of day" and "days of the week" settings to schedule this ElectricSentry instance to run.

Note: If you are setting up ElectricSentry to run in a different time zone, make sure you set the Time Zone field for the location where ElectricSentry will run.

- a. In the Advanced section, click the Enabled checkbox.
- b. Click the **Save** button.

Note: For each new schedule you create, repeat steps 6-10 above.

The Job Step Execution Environment

Terms and definitions

Various factors influence job step execution. Both a job step and its postprocessor (if any) run in the same environment, except shell. The shell used to run a command comes from what is specified in the step. If a shell is not specified on the step, the shell specified on the resource is used— this is also the same shell used for running the postprocessor.

Machine

The machine where a job step is executed is determined by the resource specified in the corresponding procedure step. If a pool is specified in the procedure step, ElectricCommander picks a specific resource from that pool. A job step can determine its actual resource by querying the property `/myResource/resourceName` or assigned `resourceName` property on the job step. The host where a step is expected is `/myResource/hostname`. If the resource is a proxy agent, this property contains the name of the proxy target. The agent host name is in `/myResource/proxyHostName`.

OS-level access control

For its resource, a job step executes under the same account as the ElectricCommander agent. You may want to contact Electric Cloud technical support for help configuring ElectricCommander agents. Basically, the agent needs to know what account it will run as, and Windows agents require additional setup if impersonation is used. If using impersonation, the job step runs under the credential effectively attached to the step.

Environment variables

For its resource, a job step inherits environment variables from the ElectricCommander agent. The agent's environment variables can be configured as part of the agent configuration. In particular, the `PATH` environment variable typically includes the ElectricCommander installation directory for easy access to applications such as *ectool* and *postp* (the ElectricCommander postprocessor).

Working directory

The default job step working directory is the top-level directory in the job's workspace. However, you can configure the working directory with a property on the procedure step. When you run on a proxy agent, the step actually runs on the proxy target, in the working directory specified in the step. If the working directory is not specified, the step runs in the UNIX path to the workspace.

Standard I/O

Standard job step output and errors are both redirected to the step log file.

ElectricCommander access

For easy access to ElectricCommander, the following environment variables are set automatically for a job step:

COMMANDER_HOME

A variable whose value is the base installation directory for an agent. On Windows, this directory is typically `C:\Program Files\Electric Cloud\ElectricCommander`. On UNIX platform, this directory is typically `/opt/electriccloud/electriccommander`.

COMMANDER_SERVER

IP address for the ElectricCommander server machine.

COMMANDER_PORT

Port number for normal communication with the ElectricCommander server.

COMMANDER_PLUGINS

(configurable) The directory where installed plugins live (for example, C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\Plugins)

COMMANDER_HTTPS_PORT

Port number for secure communication with the ElectricCommander server.

COMMANDER_JOBID

Unique identifier for the job containing the current job step.

COMMANDER_JOBSTEPID

Unique identifier for this job step.

COMMANDER_SESSIONID

ElectricCommander session identifier for the current job, which allows ectool to access ElectricCommander with job associated privileges, without an additional login.

COMMANDER_WORKSPACE_UNIX

Absolute path to the workspace directory for this job, in a form suitable for use on UNIX machines.

COMMANDER_WORKSPACE_WINDRIVE

Absolute path to the workspace directory for this job, in a form suitable for use on Windows, and starting with a drive letter.

COMMANDER_WORKSPACE_WINUNC

Absolute path to the workspace directory for this job, in a form suitable for use on Windows, specified using UNC notation.

COMMANDER_WORKSPACE

Absolute path suitable for accessing the top-level workspace directory for this job on this machine. On a UNIX machine, this has the same value as `COMMANDER_WORKSPACE_UNIX`. For Windows, it is the same as `COMMANDER_WORKSPACE_WINUNC`.

COMMANDER_WORKSPACE_NAME

This is the name of the workspace on the ElectricCommander server.

These environment variables allow you to invoke `ectool` without specifying a `--server` argument. They also provide a context for accessing properties in `ectool`. For example, "`ectool getProperty foo`" looks for the property named "foo" on the current job step. Environment variables provide `ectool` with a session including all user privileges associated with the job.

Also, if the resource is a proxy agent, these environment variables are available in the step's environment on the proxy target.

Preflight Builds

[Preflight Build Solution](#)[Preflights with Commander](#)[Samples](#)[Default SCMs](#)[Troubleshooting](#)

Why use Preflight Builds?

When developers make changes, they generally build and test their code locally and *commit* their changes if the code ran successfully. Unfortunately, this process is not sufficient.

Code changes may break the production build because of environment differences, platform-specific issues, or incomplete commits. These issues leave the product in a broken state until the changes are backed-out of the product or the developer commits a fix, which often affects the entire development team. It is a time-consuming process to find the problem when multiple developers commit their changes simultaneously and the build breaks.

Preflight Build Solution

A **Preflight Build** is used to build and test a developer's changes before those changes are committed. A "post-commit" source tree is simulated by creating a clean source snapshot and overlaying the developer's changes on top of it. These sources are then passed through the production build procedure to validate the changes work successfully.

Developers are allowed to commit their changes only if the preflight build is successful. Because developer changes are built and tested in isolation, many common reasons for broken production builds are eliminated.

Preflights with Commander

Workflow

After a preflight build is configured, the general flow of interaction between the server, agent, and client is:

- A developer invokes the client preflight program either through an IDE or via the command-line. The following arguments are specified in the XML configuration file or on the command-line:
 - Define which ElectricCommander server to use (hostname, ports, and so on),
 - Define the project, procedure, and parameters to use to start the preflight build, and
 - SCM-specific information to use (ports, changelists, and so on).
- Next, the client preflight program connects to the ElectricCommander server and launches the specified procedure.

- The job is started and the developer's changes are uploaded.
- If auto-commit was turned on, the client program waits for the job to complete. Otherwise, the client program exits immediately.
- The job progresses until it gets to the special step added to the procedure to extract sources and overlay changes.
 - The agent-side driver creates a clean source snapshot based on configuration information passed to it from the client-side.
 - The step overlays changes [uploaded by the client] on top of the snapshot to simulate the developer's check-in.
- The build continues as usual, using the modified sources.
- If the developer chose to auto-commit changes, the preflight program was waiting for the job to complete. If the job is successful, the changes are committed if:
 - No files were added or removed from the change set(s).
 - No files were modified since the preflight build started.
 - All conditions set in the driver script were met.

Components

Three components are required for an ElectricCommander preflight build:

- **Server** - This is the main Commander server that will run the preflight builds.
- **Agent** - This is any Commander agent machine that is setup as a resource for the server. This machine must be able to communicate with the source control system because it is responsible for creating a clean source snapshot on which the developer's changes are overlayed.
- **Client** - This is the machine where a developer has active code changes to submit for a preflight build. Like the agent, this machine must be able to communicate with the source control system because it is responsible for determining which files were modified so it can transmit these files to the agent.

Installation

Preflight is automatically installed with SCM plugins. Commander pre-installs numerous source code management (SCM) plugins including Accurev, Bazaar, CVS, ClearCase, Git, Mercurial, Perforce, SVN, StarTeam, Team Foundation Server, and Vault. Before you can use your preferred SCM, you will need it to communicate with the Commander server.

Configuration

This section provides details for setting up and running Preflight builds in a typical build environment.

Server

Adding the preflight snapshot step

A typical production build procedure has a step that extracts a clean source code snapshot before the actual build starts. For preflight builds, a special step needs to run in place of this snapshot step. This special step is responsible for creating the base snapshot and then overlaying the developer's changes.

- On the Procedure Details page, to create a new step click the Plugin link.
- On the Choose Step panel, select Source Code Management.
- Choose the ECSCM plugin for your SCM or from the right-pane, select the "Preflight" step for your SCM.
- On the New Step page, notice that your SCM is displayed in the Subprocedure section.
 - Set the Resource name for the step to an agent that can communicate with the SCM system, so it can create the source snapshot.
- The Parameter section also displays appropriate fields to supply information for your SCM. The following two fields are common to most SCMs:
 - Configuration - Supply the name you created for your SCM configuration.
 - Destination Directory - This is a path relative to the job's workspace, where the source tree will be created.
- Fill-in all other information required for your new step.
- Click **OK**.

Choosing between snapshot steps

Only one snapshot step should run depending on whether a production or preflight build is invoked. To enable the appropriate step, add a checkbox-style parameter to the build procedure to determine whether or not the build is a preflight. Now, use that parameter in the Run Condition field on the two snapshot steps. In the following example, the parameter is named "preflight", with an unchecked value of "false" and a checked value of "true".

- For the preflight snapshot step, set the run condition to `${preflight}`
- For the production snapshot step, set the run condition to `[/javascript getProperty("preflight") == "false"]`

Choosing how files are uploaded

By default, when developers run preflight builds, their changes are uploaded to the job workspace via the server. However, if the job workspace is on a network share, accessible to the developer's machine, a useful optimization is for files to be copied directly into the job workspace. To use the network share approach, set a flag on the top-level procedure that is invoked when running a preflight build. This property can be created on the Procedure Details web page as follows:

1. Create a nested property sheet on the procedure called `ec_preflight`.
2. Create a property in that procedure called `waitForStep` with a value of "1".
This property can be created from the command-line by calling:

```
ectool setProperty ec_preflight/waitForStep 1 --projectName <project> --
procedureName <procedure>
```

Agent

The agent machine must be able to communicate with the SCM system to create the source snapshot.

Client

From a developer's machine, there are two ways to start preflight builds:

- From an IDE:
 - Eclipse
Open a Run dialogue, select Launch Commander Procedure, then select an existing preflight configuration or create a new one.
For more information, see the *ElectricCommander Eclipse Integration Tech Note* PDF file or the Eclipse plugin.
 - Visual Studio (See the *ElectricCommander Visual Studio Tech Note* PDF file or the Visual Studio plugin)
- From the command line:
The executable program, `ecclientpreflight`, is used to run a preflight build from the command-line. This program is included in the ElectricCommander "Tools" installation. When you run `ecclientpreflight`, configuration options can be passed in an XML configuration file or on the command-line.

If the same option is specified in both the configuration file and on the command-line, the value passed on the command-line takes precedence. The XML configuration file can be passed to the program via the `--config` option. If the XML configuration file is not passed in, current and ancestor directories are searched for a file named `".preflight"`, using the first one it finds as the configuration file.

Generally, a `".preflight"` file is stored at the root of a developer's workspace, containing configuration information specific to that workspace.

Samples

In the following samples, the SCM was set to Perforce.

The following table is output from `"ecclientpreflight --help"`, enumerating the command-line options. Some options may be specified using an abbreviated format (`-c`, with a single dash) or a more verbose format (`--config`, with two dashes).

Electric Cloud Client-Side Preflight Tool, Version: 4.x Copyright (C) 2012 Electric Cloud, Inc. All rights reserved. Usage: <code>ecclientpreflight [options]</code>	
General Options	
<code>-c, --config <file></code>	Load configuration options from the specified XML file. If <code><file></code> is <code>'-'</code> , read standard input. If not supplied, the current and ancestor directories will be searched for a <code>.preflight</code> file. The first <code>.preflight</code> found will be used as the preflight configuration.
<code>--load <file></code>	Evaluate <code><file></code> after option processing. May appear multiple times.
<code>-d, --debug</code>	Display debugging information.
<code>-v, --version</code>	Display version information.

<code>-h, --help</code>	Display this information.
Server Communication Options	
<code>--server <name></code>	The host name (defaults to localhost).
<code>--port <port></code>	The HTTP port (defaults to 8000).
<code>--securePort <port></code>	The HTTPS port (defaults to 8443).
<code>--secure</code>	If set, only secure connections are used. Off, by default.
<code>--timeout <timeout></code>	The response timeout in seconds (defaults to 180).
<code>--userName <name></code>	The name of the ElectricCommander user to login as.
<code>--password <password></code>	The password for the specified user. If blank and the user has no active session, a prompt to enter the password will be provided.
<code>--driverLocation</code>	Property path to a property sheet containing the driver scripts that will be loaded and run after a connection to the server is established. Defaults to <code>/server/ec_preflight</code> .
<code>--mainDriver</code>	The name of a property in the sheet specified by <code>driverLocation</code> . This is the main driver script that will be loaded and run after a connection to the server is established.
Logging Options	
<code>-l, --log</code>	If specified, debug information is logged. Off, by default.
<code>--logDir</code>	Where to store log and other files. Defaults to the user's home directory.
Procedure Invocation Options	
<code>--projectName <name></code>	The name of the ElectricCommander project containing the procedure to invoke.
<code>--procedureName <name></code>	The name of the ElectricCommander procedure to invoke.

<code>-p, --param <name>=<value></code>	Supply additional parameters to the procedure. May appear multiple times. If any parameters were specified in the config file, those supplied on the command-line append to that list, overriding parameters with the same name.
<code>--priority</code>	The priority of the job. Possible values are low, normal, high, highest. If left unspecified, defaults to normal.
<code>--jobTimeout <timeout></code>	The number of seconds to wait for the job to complete when auto-committing changes. Defaults to 3600 seconds (1 hour).
<code>--waitForJob</code>	Wait for the job to complete and report its outcome. This action does not occur by default unless a set of SCM charges are being committed automatically.
<code>--runOnly</code>	Run the procedure and exit immediately. The SCM driver is not downloaded in this case.
SCM Options	
<code>--scmType</code>	<p>The name of the SCM, required, unless the procedure is invoked in 'run only' mode. The driver is downloaded from <code>\$driverLocation/clientDrivers/\$scmType</code>.</p> <p>Valid values: ECSCM-Accurev, ECSCM-Bazaar, ECSCM-ClearCase, ECSCM-CVS, ECSCM-Git, ECSCM-Mercurial, ECSCM-Perforce, ECSCM-Repo, ECSCM-StarTeam, ECSCM-SVN, ECSCM-TFS, ECSCM-Vault.</p>
<code>--autoCommit <1 0></code>	Whether or not the changes should be automatically committed if the job completes successfully. Off, by default.
<code>--commitComment <comment></code>	A comment for the auto-commit.
Perforce Options	
<code>--p4port <port></code>	The value of P4PORT. May also be set in the environment. This is a required value.
<code>--p4user <user></code>	The value of P4USER. May also be set in the environment. This is a required value.
<code>--p4passwd <password></code>	The value of P4PASSWD. May also be set in the environment.

<code>--p4client <client></code>	The value of P4CLIENT. May also be set in the environment. This is a required value.
<code>--p4template <template></code>	The name of a Perforce client used to create a base snapshot before overlaying local changes. Defaults to the value of <code>--p4client</code> if not specified.
<code>--p4changelist <change></code>	The changelist number (or default) whose changes are being tested. May be specified multiple times. If no changelists are specified, all client changelists will be used.

The following is a sample XML configuration file with all options specified:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
  <server>
    <userName>myUser</userName>
    <password>myPass</password>
    <hostName>commanderServer</hostName>
    <port>1234</port>
    <securePort>2345</securePort>
    <stompPort>3456</stompPort>
    <secure>0</secure>
    <timeout>3456</timeout>
    <driverLocation>/server/ec_preflight/alternateDrivers</driverLocation>
    <mainDriver>myMainClientDriver</mainDriver>
  </server>
  <procedure>
    <projectName>myProject</projectName>
    <procedureName>myProcedure</procedureName>
    <parameter>
      <name>branch</name>
      <value>main</value>
    </parameter>
    <parameter>
      <name>preflight</name>
      <value>true</value>
    </parameter>
  </procedure>
</data>
```

```
        <priority>high</priority>
        <waitForJob>1</waitForJob>
        <jobTimeout>3600</jobTimeout>
    </procedure>
    <scm>
        <type>perforce</type>
        <port>perf:1234</port>
        <user>myUser</user>
        <password>myPass</password>
        <client>myUser-main-client</client>
        <template>myUser-main-template</template>
        <changelist>default</changelist>
        <changelist>67382</changelist>
        <autoCommit>1</autoCommit>
        <commitComment>Fixing bug 38582.</commitComment>
    </scm>
</data>
```

In most cases, a developer does not need to override all default values, and will probably want to specify passwords and some SCM-specific options on the command-line.

The following is a more typical XML configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<data>
    <server>
        <userName>myUser</userName>
        <hostName>commanderServer</hostName>
    </server>
    <procedure>
        <projectName>myProject</projectName>
        <procedureName>myProcedure</procedureName>
        <parameter>
            <name>branch</name>
            <value>main</value>
        </parameter>
        <parameter>
            <name>preflight</name>
            <value>true</value>
        </parameter>
    </procedure>
</data>
```

```

    </parameter>

    <jobTimeout>3600</jobTimeout>
  </procedure>

  <scm>

    <type>perforce</type>

    <port>perf:1234</port>

    <user>myUser</user>

    <client>myUser-main-client</client>

  </scm>

</data>

```

If this file is stored in a .preflight file at the top-level of a source tree, the command to start the preflight would look something like this:

```
ecclientpreflight --p4changelist 56793 --autoCommit 1 --commitComment "Fixing
bug 38582."
```

Default SCMs

ElectricCommander includes preflight support for some of the most common Source Control (SCM) systems. For an SCM plugin you installed, look for a Help link for that plugin (on the Plugins page). Command-line options are included in this help topic for the following SCMs:

Acc uR ev	Git	Star Tea m
Baz aar	Me rcu rial	Sub vers ion
Cle arC ase	Pe rfo rce	TFS
CV S	Re po	Vau lt

Perforce

The command-line options for the Perforce driver for `ecclientpreflight` are:

Perforce Options / Descriptions	
<code>--p4port <port></code>	Required: The value of P4PORT may be set in the environment also.
<code>--p4user <user></code>	Required: The value of P4USER may be set in the environment also.
<code>--p4passwd <password></code>	The value of P4PASSWD may be set in the environment also.
<code>--p4client <client></code>	Required: The value of P4CLIENT may be set in the environment also.
<code>--p4template <template></code>	The name of a Perforce client used to create a base snapshot before overlaying local changes. Defaults to the value of <code>--p4client</code> if not specified.
<code>--p4changelist <change></code>	The changelist number (or default) whose changes are being tested and can be specified multiple times. If no changelists are specified, all client changelists will be used.
<code>--p4synctochange <change></code>	The changelist number the Preflight Job should use when sync'ing the source tree. Values are: <i>head</i> - The most recent changelist anywhere in the P4 depot (default). <i>have</i> - The changelist for the most recent file that was synced to 'p4client' changelist. This is a p4changelist number.

Subversion

The command-line options for the Subversion driver for `ecclientpreflight` are:

Subversion Options / Descriptions	
<code>--svnpath <path></code>	The path to the locally accessible source directory in which changes were made.
<code>--svnupdatetohead</code>	Use this option to update the agent workspace created during preflight -- updated to HEAD. By default, the agent workspace is updated to the revision found in the client workspace.
<code>--svnignoreexternals</code>	Causes the preflight process to ignore svn externals.

AccuRev

The command-line options for the AccuRev driver for `ecclientpreflight` are:

AccuRev Options / Descriptions	
<code>--accurevuser <user></code>	The value of ACCUREVUSER may be set in the environment also. If not specified, the default is to the Commander user.
<code>--accurevpasswd <password></code>	Required: The value of ACCUREVPASSWD may be set in the environment also. This is a required value.
<code>--accurevpending <pending></code>	Use this option to scan the client workspace for all pending elements using the "stat -fn -p" command. By default, the workspace is scanned for all kept elements using the "stat -fn -k" command.
<code>--accurevpath <path></code>	Required: The value of the ACCUREVPATH. Also may be set in the environment.

ClearCase

Preflight support for ClearCase is available only for snapshot views. The command-line options for the ClearCase driver for `ecclientpreflight` are:

ClearCase Options / Descriptions	
<code>--ccpath <path></code>	The path to the locally accessible source directory where changes were made.
<code>--ccUnixCSpecPath <unixCSpecPath></code>	The path to the appropriate config spec on a UNIX platform for the current view.
<code>--ccWinCSpecPath <winCSpecPath></code>	The path to the appropriate config spec on a Windows platform for the current view.
<code>--ccUnixRelativePath <unixRelativePath></code>	The relative path for the current view [on a UNIX platform] from the view root to the directory where changes were made.
<code>--ccWinRelativePath <winRelativePath></code>	The relative path for the current view [on a Windows platform] from the view root to the directory where changes were made.

Bazaar

The command-line options for the Bazaar driver for `ecclientpreflight` are:

Bazaar Options / Descriptions	
<code>--workdir <path></code>	The developer's source directory.
<code>--branch <name></code>	Branch name for the preflight.
<code>--method=<local remote></code>	<code>local</code> - get tracked and untracked changes in the current workingdir <code>remote</code> - get changes between working tree and remote branch

Git

The command-line options for the Git driver for `ecclientpreflight` are:

Git Options / Descriptions	
<code>--gitdir=dir</code>	The Git directory to process.
<code>--method=<local_all local_tracked remote></code>	<ul style="list-style-type: none"> <code>local_all</code> - get tracked and untracked changes between working tree and local repo <code>local_tracked</code> - get tracked changes between working tree and local repo <code>remote</code>

CVS

The command-line options for the CVS driver for `ecclientpreflight` are:

CVS Options / Descriptions	
<code>--cvsroot <path></code>	The path to the repository.
<code>--module <module></code>	The module name for the preflight.
<code>--workdir <path></code>	The developer's source directory.

Mercurial

The command-line options for the Mercurial driver for `ecclientpreflight` are:

Mercurial Options / Descriptions	
<code>--hgpath <path></code>	The path to the locally accessible source directory in which changes were made. Generally, this path is to the root of the workspace.

Repo

The command-line options for the Repo driver for `ecclientpreflight` are:

Repo Options / Descriptions	
<code>--repoworkdir <path></code>	The developer's source directory.
<code>--agentworkdir <path></code>	The path to the source directory, used by the agent.

StarTeam

The command-line options for the StarTeam driver for `ecclientpreflight` are:

StarTeam Options / Descriptions	
<code>--STProjectName <project></code>	The StarTeam project name.
<code>--workingdir <path></code>	Working dir for the updated files.

TFS

The command-line options for the Team Foundation Server driver for `ecclientpreflight` are:

TFS Options / Descriptions	
<code>--server <url></code>	The equivalent value to Collection in VS2008 and under. If you use this option, it is assumed you are using TF version VS2008 or under and it will use the option <code>/server</code> in the preflight. This field is required if you have VS2008.
<code>--collection <url></code>	The URL that points to <code>/collection</code> . This value is used for VS2010. If you specify this value, the command uses the option <code>/collection</code> when it executes the TF query for preflight. This field is required if you have VS2010.
<code>--localfolder</code>	The path to the locally accessible source directory in which changes were made. Generally, this is the path to the root of the workspace.
<code>--workspace <workspace></code>	The workspace containing the data for the preflight.

Vault

The command-line options for the Vault driver for `ecclientpreflight` are:

Vault Options / Descriptions	
<code>--workingdir <path></code>	Working dir for the updated files.

Troubleshooting

Client

The `--debug` option to `ecclientpreflight` provides additional information about internal actions taken when starting the preflight build. By modifying the properties containing driver scripts, debug output can be added where needed. If you want to capture debug information to a log file, use the `--log` option. By default, logs are stored in the user's home directory. To change this location, use the `--logDir` option.

Agent

The step log for the custom preflight step provides information about the snapshot being created and the files being overlayed. Debug output can be added where needed by modifying the properties containing the driver scripts

Properties

[Creating or modifying properties](#)

[Using property values](#)

[Property sheets and intrinsic properties](#)

[Property names and paths](#)

[Context-relative "shortcuts" to property paths](#)

[Property path shortcuts in ElectricCommander 5.0 and later](#)

[Property name substitutions](#)

[Expandable properties](#)

[The property hierarchy](#)

[Special property references](#)

[Intrinsic properties listed by object type](#)

[Commander Object / Property tables](#)

Overview

ElectricCommander provides a powerful data model based around the notion of a *property*.

- A property is a string value with a name.
- Properties can have arbitrary names and values.
- You can attach properties to any object in the Commander system, such as a project, procedure, or job.
- After a property is created, it is stored in the Commander database.
- You can retrieve and/or modify the property value later.
- Properties you no longer need can be deleted.

Properties are used extensively throughout the Commander system and provide a flexible and powerful mechanism to manage data about your builds.

Property Use Case Examples

- When a job starts, it computes a unique identifier for that build and saves it in a property. Later build steps can retrieve the property value to embed the build number in binaries generated during the job.
- When a build executes, it can set a property on the procedure to identify the source code version used for the build (for example, a tag or timestamp from your source code control system). The next time the build executes, it can retrieve the property value from the procedure and use it to extract information from your source code control system about the files that changed since the last run.

- Suppose you have a collection of machines for testing use, and some machines have older test hardware and some machines have newer hardware with slightly different characteristics. You can set a property on each test machine resource to indicate which version of test hardware is available on which machine. Later, when a test executes, the test can retrieve the property for the machine where it ran and configure tests appropriately for that hardware. If you upgrade test hardware on a machine, all you need to do is change the property on that resource to reflect the new version.
- You can set job properties to indicate the build status produced by that job. For example, if your QA team finds fatal flaws in a build, it can mark the job accordingly. Builds that need to be preserved (release candidates or builds undergoing beta testing at customer sites) can be marked with properties so those builds are not deleted.
- When a job executes, properties are set for its job steps that hold metrics such as how many files compiled during a build step, how many tests executed during a test step, or how many errors were detected in the step. These property values are included in reports and can be examined later to compute trends over time. You can define additional properties for metrics useful to you.

Commander provides Intrinsic properties and allows you to create Custom properties. This help topic enumerates properties available within Commander (Intrinsic properties), distinguishes between relative and absolute property paths, and describes property hierarchies.

Note: Intrinsic properties are case-sensitive. Custom properties, like all other object names in the Commander system, are "case-preserving," but *not case-sensitive*.

- **Intrinsic properties**

These properties represent attributes that describe the object to which they are attached, and are provided automatically by Commander for each similar type object. For example, every project has a `Description` property that can be referenced with a non-local property path such as `/projects/Examples/description`.

- **Custom properties**

Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are *not* created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.

Creating or modifying properties

You can set a property value by using one of the following three methods or applications: the Commander web interface, the **ectool** command-line application, or the Perl API.

- Using the web interface, you will see a table labeled Custom Properties on the pages that display details for projects, procedures, and so on.
Click on the **Create New Property** action to create a new property for that object, or click on an existing property to change its value.
- Using the ectool application, set a property value with a command like:

```
ectool setProperty /myJob/installStatus complete
```

This command sets the property named `/myJob/installStatus` to the value `complete`, and creates the property if it did not already exist (the meaning of property names like

/myJob/installStatus is explained below). You can use ectool from within one job step to set properties accessed by later steps in the same job.

- Using the Perl API, you can use an external script or a script in a step with *ec-perl* as the shell. A Perl code example:

```
use ElectricCommander ();
my $ec = new ElectricCommander;
$ec->setProperty ("/myJob/installStatus", "complete", {jobStepId =>
    $ENV{COMMANDER_JOBSTEPID})
```

Note: Using the Perl API may yield better performance if you are requesting multiple API calls in one step.

For more information, review the [Using the Commander API](#) Help topic.

Using property values

A job step can access a property value by two methods. The first method is *substitution*, using the `$()` notation. Suppose you enter the following text as a command for a step:

```
make PLATFORM=$(platform)
```

Before running the step, Commander finds the property named `platform` and substitutes its value in the command string in place of `$(platform)`. For example, if the property contains the value "windows", the actual command executed is:

```
make PLATFORM=windows
```

The substitution method can be used for a step's command and for any step fields, such as its resource or its working directory. This method allows you to compute configuration information in an early step of a job, then use that configuration information to control later steps in the job.

The second method for a step to access the property value is with ectool's `getProperty` command. For example, the following command returns the property value named `platform`:

```
ectool getProperty platform
```

Property sheets and intrinsic properties

A property value can be a simple string or a nested *property sheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.

Most objects have an associated "property sheet" that contains "custom properties" created by user scripts. The property sheet is an intrinsic property of the containing object called "property sheet", so to reference a project's custom property "branchName", you could specify `/projects/aProject/propertySheet/branchName`.

As a convenience, Commander allows the "property sheet" path element to be omitted and the path written as `/projects/aProject/branchName`. If there is no intrinsic property with the same name, the path will find the property on the property sheet.

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 via the property sheet property of their containing sheet.

For example:

```
/projects/aProject/propertySheet/topSheet/propertySheet/propB
- or -
/projects/aProject/topSheet/propB
```

All information managed by ElectricCommander exists in the form of properties and property sheets and your own custom-created properties. For example, each project, procedure, and step is represented internally as a property sheet—the command for a step is actually a property associated with the step, and so on. Every value in the Commander system can be accessed as a property, using the naming facilities described below. These properties are called *intrinsic* properties. **Commander enforces some restrictions on intrinsic property values, whereas custom properties can have any value you choose.**

To learn more about intrinsic properties defined for an object by ElectricCommander, consult the [ectool](#) and the [Commander API](#) help topic. For example, to learn about intrinsic properties associated with each project, find the documentation for the `getProject` ectool command. The result of running this command is an XML document whose field names and values represent the properties for the project.

Note: At the end of this help topic, find the current list of Commander intrinsic properties sorted by each object and its associated properties [in table format].

Property names and paths

Properties are named using multi-level paths such as `first/second/third`, which refers to a property named "third" in a property sheet named "second" in a property sheet named "first." ElectricCommander also supports an equivalent notation using brackets instead of slashes. For example, `first[second]/third` and `first[second][third]` both refer to the same property as `first/second/third`. The bracket `[]` notation is based on matching brackets. For example, `steps[build[1]]` refers to a property named "build[1]" inside a property sheet named "steps" (it does NOT treat "build" as a property sheet containing a property named "1").

Note: Slashes are not considered separators when they appear between brackets.

Property names/paths have two forms: *absolute* and *relative*.

Absolute property paths

Absolute property paths are referenced by a fully-qualified path syntax that begins with a slash character ("/") followed by a top-level name. This path syntax is similar to a file system path specification. The first component after the "/" must be one of several reserved words that select a starting location to look up the property name. For example, consider the property name `/server/Electric Cloud/installDirectory`. `/server` means "lookup" starts in the topmost property sheet associated with the Commander server. This property name refers to the "installDirectory" property inside the server property sheet.

The system defines the following top-level names (absolute property names):

```
/artifacts/...
```

Start in the property sheet containing all artifacts. For example, `/artifacts/myGrp:myKey/prop1` refers to the `prop1` property on the artifact whose name is "myGrp:myKey".

```
/artifactVersions/...
```

Start in the property sheet containing all artifact versions. For example, `/artifactVersions/myGrp:myKey:1.0-36/prop1` refers to the `prop1` property on the artifact version whose name is `"myGrp:myKey:1.0-36"`. **Note:** Throughout the API, you can substitute `"groupId:artifactKey:version"` wherever an `artifactVersionName` argument can be specified. This is the same if the `artifactVersionNameTemplate` specified in the artifact is in the form `"groupId:artifactKey:version"`. For cases when the template is different, it is convenient to be able to specify this tuple if you do not know the `artifactVersionName`.

`/groups/...`

Start in the property sheet containing all groups. For example, `/groups[dev]` refers to the group named `"dev"`.

`/projects/...`

Start in the property sheet containing all projects. For example, `/projects[nightly]` refers to the project named `"nightly,"` and `/projects[nightly]/procedures[main]` refers to a procedure named `"main"` in that project.

`/jobs/...`

Start in the property sheet containing all jobs. For example, `/jobs[ecloud.4096]` refers to the job named `"ecloud.4096."` You can name a job using either its name (as in the preceding example) or using the unique identifier assigned to it by Commander.

`/plugins/...`

Start in the property sheet containing all plugins. For example, `/plugins[EC-AgentManagement]` refers to the currently promoted plugin named `"EC-AgentManagement"` and `/plugins[EC-AgentManagement]/project/procedures[scpCopyFile]` refers to a procedure named `"scpCopyFile"` in that plugin.

Note: There is a subtle difference between

```
ectool setProperty /plugins/EC-AgentManagement/project/foo 'bar'
```

and

```
ectool setProperty /plugins/EC-AgentManagement/foo 'bar'
```

The former places the property on the plugin's project and can be referenced by `$/myProject/foo` while the latter places the property on the plugin object and will not be found via `$/myPlugin/foo`.

`/repositories/...`

Start in the property sheet containing all artifact repositories. For example, `/repositories/repo1/prop1` refers to the `prop1` property on the repository whose name is `"repo1"`.

`/resourcePools/...`

Start in the property sheet containing all resource pools. For example, `/resourcePools[linuxA]` refers to the resource pool named `"linuxA"`.

`/resources/...`

Start in the property sheet containing all resources. For example, `/resources[linux1]` refers to the resource named "linux1".

`/server/...`

Start in the top-level server property sheet. For example, `/server/a` refers to the server property named "a".

`/users/...`

Start in the property sheet containing all users. For example, `/users[Bob]` refers to the user named "Bob".

`/workspaces/...`

Start in the property sheet containing all workspaces. For example, `/workspaces[default]` refers to the workspace named "default."

Relative property paths

Property names that do not begin with "/" are *relative* and looked up starting in the *current context*. For example, when executing a job step, the current context includes properties defined for the job step, parameters for the current procedure, and global properties on the current job.

Relative property paths are distinguished from absolute property paths because they do not begin with one of the top-level names. To avoid having to construct full property paths, Commander supports the concept of a relative property path.

In use, the relative property path value is resolved by its context and a defined search order, which results in accessing the value of an absolute fully-specified property value. Contexts and search orders are as follows:

1. In a job step context, Commander searches for relative property paths in the following order:
 - a property on the job step object
 - a property of the parent job step object, which includes parameters of the procedure on which the step is defined
 - a property on the job object

Notes:

- The search can be enabled or disabled by using the `--extendedContextSearch` option on the `ectool getProperty` or `setProperty` commands. When searching for a property value, disable the search by setting the `--extendedContextSearch` switch to "false", requiring the property to exist on the job step object or return false. When writing a new value to a property, enable the search by setting the `--extendedContextSearch` switch to "true", allowing the search for a property within the search order before creating a new one. New properties are created on the job step object.
 - Parameters for subprocedure calls from job steps are searched for in a job step context.
 - For procedure parameters for nested subprocedures, properties referenced by parameters are looked up as described [above] for job steps.
2. When expanding schedule parameters, Commander searches the relative property path in the following order:

- a property on the procedure being called
 - a property on the project on which the procedure being called is defined
 - a property on the server
3. In a job name template context, Commander searches for the relative property path as a property on the job.
 4. In any other context, Commander searches for the relative property path as a property on the context object.

Context-relative "shortcuts" to property paths

A 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. 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.

Shortcuts to property paths that provide convenient runtime access include:

`/myArtifactVersion/...`

Start in the property sheet for the `artifactVersion` associated with the current context. The only context where this property has any value is in the `artifactVersionNameTemplate` field for the artifact.

`/myCall/...`

`/myCall/` is now a subset of `/myParent/`. (See the `/myParent/` description below.) `/myCall/` refers to the parent (job or jobStep's) CUSTOM property sheet, so you can reference custom properties and parameters only, which get copied into the custom property sheet. `/myJob/`, on the other hand, points directly to the job object, so you can reference intrinsic job properties in addition to the properties in the custom property sheet.

`/myCredential/...`

Start in the property sheet for the credential associated with the current job step. This form produces an error if the current job step does not have a credential.

`/myEvent/...`

This is a special property used only within an "email notifier." `/myEvent/` allows you to refer to fields associated with the notifier itself. You can include these properties, for example, in the text of the email you send out for notification:

`/myEvent/notifier` - This property contains the name of the notifier that created the notifier event. You created this name when you created the notifier.

`/myEvent/entity` - This property contains the object where the notifier is attached. Two possible values are "job" or "jobStep," depending on where the notifier is attached.

`/myEvent/source` - This property contains the name of either the job or the jobStep where the notifier is attached.

`/myEvent/type` - This property defines whether the notifier is an "On Start" or an "On Completion" notifier. Two possible values are "STARTED" or "COMPLETED".

`/myEvent/time` - This property contains the time when the notifier occurred. The time is always specified in GMT and uses a formatted string, for example,
2009-06-11T21:00:56.502Z

`/myEvent/timeMillis` - This property contains the "timestamp" in milliseconds when the notifier occurred. This value is the number of milliseconds since January 1, 1970 GMT. The `timeMillis` corresponding to the time in the previous example is: 1244754056502

`/myJob/...`

Start in the global property sheet for the current job. `/myJob/` points directly to the job object so you can reference built-in job properties (`jobName`, `createTime`, `outcome`, and so on) as well as custom properties. Also, this property sheet can be used to hold working data that needs to be passed from one step to another within the job.

`/myJobStep/...`

Start in the property sheet for the current job step.

`/myParent/...`

Start in the global property sheet for the parent job step or job if this is a top-level step. `/myParent/` points directly to the job or job step object, so you can reference intrinsic or custom properties. For example, you can reference the parameters that were passed to this procedure.

In addition, you can reference a sibling step by calling `/myParent/jobSteps/<sibling step name>`.

Or, you can create additional properties on `/myParent/` to pass information from one step in the procedure to another:

step1 calls `setProperty /myParent/results 100`

step2 can call `getProperty /myParent/results`

`/myParent/` is a superset of `/myCall/`. You can continue to use `/myCall/`, but Electric Cloud recommends using `/myParent/` in new scripts you create.

`/myProcedure/...`

Start in the property sheet for the procedure in which the current job step was defined. If the current job step is executing as part of a nested procedure, `/myProcedure/` refers to the innermost nested procedure.

`/myProject/...`

Start in the property sheet for the project in which the current job step was defined. In the case of nested procedure invocations, this is the project associated with the innermost nested procedure, for example, the project associated with `/myProcedure/`.

`/myResource/...`

Start in the property sheet for the resource assigned to the current job step.

`/myResourcePool/...`

Start in the property sheet for the resource pool that provided the resource for the current job step. Returns null if the step did not specify a resource pool.

`/myStep/...`

Start in the property sheet for the current step. "Step" refers to the (static) definition of a step, which is part of a procedure— this is different from a job step, which represents a step when it executes (dynamically) in a job. Use `/myJobStep/` to access the job step.

`/myState/...`

Start in the property sheet for the state object so you can reference built-in and custom state properties or find parameter values passed to that state. When accessed from a state, `/myState/` refers to that state. When accessed from a transition, `/myState` refers to the transition's owning state. When accessed from a job or job step, `/myState/` refers to the state that launched that job as a subjob.

`/mySubjob/...`

Start in the property sheet for the subjob so you can reference built-in and custom job properties, parameters passed to the job, and properties on steps within that job. When accessed from a transition, `/mySubjob/` refers to the subjob started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the outcome or other information for the subjob can influence which state the workflow transitions to next.

`/mySubworkflow/...`

Start in the property sheet for the subworkflow so you can reference built-in and custom workflow properties. When accessed from a transition, `/mySubworkflow/` refers to the subworkflow started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the active state and other information for the subworkflow can influence which state the workflow transitions to next. You can access information about states and transitions belonging to the workflow by using the path `/mySubworkflow/states/someState` or `/mySubworkflow/states/someState/transitions/someTransition`.

`/myTransition/...`

Start in the property sheet for the transition object so you can reference intrinsic and custom transition properties. `/myTransition/` is accessible only from a transition.

`/myUser/...`

This property can be used only if the current session is associated with: the predefined "admin" user, a user defined as "local", or a user defined by a Directory Provider (LDAP or ActiveDirectory). This property cannot be used if the user is a "project principal", which is normally the case when running inside a Commander step.

For example, with an interactive login you can use:

```
ectool getProperty /myUser/userName (to get the user name of the logged in user, or...)
ectool getProperty /myUser/email (to get the email address)
```

`/myWorkflow/...`

Start in the property sheet for the workflow object so you can reference built-in and custom workflow properties. When accessed from a state or transition, `/myWorkflow/` refers to the "owning" workflow. When accessed from a job or job step, `/myWorkflow/` refers to the workflow whose state launched that job as a subjob. You can access information about states and transitions belonging to the workflow by using the path `/myWorkflow/states/someState` or `/myWorkflow/states/someState/transitions/someTransition`.

`/myWorkspace/...`

Start in the property sheet for the workspace associated with the current job step.

Property path shortcuts in ElectricCommander 5.0 and later

The following shortcuts to property paths are available in ElectricCommander and later. For more information, go to [Context-relative "shortcuts" to property paths](#).

Shortcuts	Descriptions	For jobs	For job steps
<code>/myApplication/...</code>	Starts in the property sheet for the application associated with the current job or job step.	Yes	Yes
<code>/myApplicationTier/...</code>	Starts in the property sheet for the application tier associated with the current job step.	No	Yes
<code>/myCall/...</code>	Starts in the property sheet for the call associated with the current job step.	No	Yes
<code>/myComponent/...</code>	Starts with the property sheet for the component associated with the current job step.	No	Yes
<code>/myCredential/...</code>	Starts with the property sheet for the credential associated with the current job step.	No	Yes
<code>/myEnvironment/...</code>	Starts in the property sheet for the environment associated with the current job or job step.	Yes	Yes
<code>/myEnvironmentTier/...</code>	Starts in the property sheet for the environment tier associated with the current job step.	No	Yes
<code>/myJob/...</code>	Starts in the property sheet for the job associated with the current job or job step.	Yes	Yes
<code>/myJobStep/...</code>	Starts in the property sheet for the job step associated with the current job step.	No	Yes

Shortcuts	Descriptions	For jobs	For job steps
/myParent/...	Starts in the global property sheet for the parent job step.	No	Yes
/myPlugin/...	Starts in the property sheet for the plugin associated with the current job.	Yes	No
/myProcedure/...	Starts in the property sheet for the procedure in which the current job or job step was defined.	Yes	Yes
/myProcess/...	Starts in the property sheet for the process in which the current job or job step was defined.	Yes	Yes
/myProcessStep/...	Starts in the property sheet for the process step in which the current job step was defined.	No	Yes
/myProject/...	Starts in the property sheet for the project in which the current job was defined.	Yes	No
/myResource/...	Starts in the property sheet for the resource associated with the current job step.	No	Yes
/myResourcePool/...	Starts in the property sheet for the resource pool associated with the current job step.	No	Yes
/myState/...	Starts in the property sheet for the state object so you can reference built-in and custom state properties or find parameter values passed to that state.	Yes	Yes
/myStep/...	Starts in the property sheet for the step associated with the current job step.	No	Yes
/myWorkflow/...	Starts in the property sheet for the workflow object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkflowDefinition/...	Starts in the property sheet for the workflow definition object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkspace/...	Starts in the property sheet for the workspace associated with the current job step.	No	Yes

Property name substitutions

Property names can contain references to other properties, which are then substituted into the property name before looking it up. For example, consider the following property name:

```
/myStep/${/myProcedure/name}
```

If the value of `/myProcedure/name` is "xyz", the property above is equivalent to `/myStep/xyz`.

Expandable properties

Property values can contain property references using the `"${}"` notation.

For example:

1. Create a property named "foo" with a value of `hello ${bar}`.
2. Create a property named "bar" with a value of `world`.
3. Reference "foo" (either using `${foo}` or `ectool getProperty foo`). The value "hello world" is returned.

If you want just the literal value of "foo" (useful in the UI, for example), you can use the `expand` option in `ectool`:

```
ectool getProperty foo --expand false. The value "hello ${bar}" will be returned.
```

Properties are expanded by default when you use `getProperty` or `getProperties`.

If the value of a property contains `"${}"` but you do not want it to be interpreted as a property reference, you can use the `expandable` option:

```
ectool setProperty symbols '${!@}#' --expandable false.
```

This option can be toggled in the web UI as well. Properties are expandable by default.

Because you cannot control where your expandable property might be referenced (and therefore which context is used during expansion), Electric Cloud recommends using absolute paths when referencing a property from the value of another property.

In the example above, if you define both "foo" and "bar" as properties on a project "proj1", you might assume there is no problem with the value of "foo". However, if you later reference "foo" from a job under "proj1" (for example, `${/myProject/foo}`), foo will be referenced with the job step as its context. Therefore, when the value of "foo" is expanded, you will get a `PROPERTY_REFERENCE_ERROR` because "bar" is not defined in the context of the job step.

Custom property names and values

The following properties are used by the standard ElectricCommander UI, so you should use these property names whenever possible, and avoid using these names in ways that conflict with the definitions below.

- **preSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *before* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

- **postSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *after* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **summary** - if this property exists, its value is displayed in the "Status" field (in the job reports) for this step, replacing whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

The property hierarchy

All ElectricCommander properties fall into a hierarchical structure, and you can reference any property using an absolute path from the root of the hierarchy. This includes properties you define and intrinsic properties defined by Commander.

For example, each step contains a property "resource" that provides the resource name to use for that step. All steps of a procedure exist as property sheets underneath the procedure. All procedures in a project exist as property sheets underneath the project, and so on.

The examples below illustrate some nesting relationships between objects.

For example, the notation "*project/procedures[procedureName]*" means each project object contains a property sheet named "procedures" holding all procedures defined within that project. Within the procedures property sheet, there is a nested property sheet for each procedure, named after the procedure. Thus, the name "*/projects[a]/procedures[b]*" refers to a procedure named "b" contained in a project named "a."

- Each project contains procedures, schedules, credential definitions, workflow definitions, and workflows:

```
project/procedures[procedureName]  
project/schedules[scheduleName]  
project/credentials[credentialName]  
project/workflowDefinitions[workflowName]  
project/workflows[workflowName]
```

- Each procedure contains steps and parameters. The parameters are "formal parameters," meaning they specify parameter names the procedure will accept, whether each parameter is required, and so on:

```
procedure/steps[stepName]  
procedure/formalParameters[paramenterName]
```

- Steps that invoke subprocedures contain parameter values for the subprocedure. These are called "actual parameters" because they provide actual values that will be passed into the subprocedure:
step/actualParameters[parameterName]

- Each job contains a collection of job step objects for steps in the outermost procedure, along with parameter values passed into the job when it was invoked:

```
job/jobSteps[stepName]  
job/actualParameters[parameterName]
```

- If a job step invokes a nested subprocedure, its property sheet contains parameter values that were passed into the nested subprocedure, plus all job steps corresponding to that procedure:

```
jobStep/actualParameters[parameterName]  
jobStep/jobSteps[stepName]
```


- Schedules contain parameter values for the procedures they invoke. These are called "actual parameters" because they provide actual values passed into the procedure:
`schedule/actualParameters[parameterName]`
- Workflow definitions contain state definitions:
`workflowDefinition/stateDefinitions[stateDefinitionName]`
- Transition definitions contain actual parameters:
`transitionDefinition/actualParameters[parameterName]`
- State definitions contain transition definitions, actual parameters, and formal parameters:
`stateDefinition/actualParameters[parameterName]`
`stateDefinition/formalParameters[parameterName]`
`stateDefinition/transitionDefinitions[transitionDefinitionName]`
- Workflows contains states:
`workflow/states[stateName]`
- Transitions contain actual parameters:
`transition/actualParameters[parameterName]`
- States contain transitions, actual parameters, and formal parameters:
`state/actualParameters[parameterName]`
`state/formalParameters[parameterName]`
`state/transitions[transitionName]`

Special property references

ElectricCommander also supports several special property reference forms that are described in the following subsections.

increment

Use this form to increment the value of an integer property before returning its value. For example, suppose property xyz has the value 43. The property reference `$[/increment xyz]` first increments the value of property xyz to 44, then returns 44.

timestamp

Use this form to generate a formatted timestamp value. For example, the property reference `$[/timestamp yyyy-MM-d hh:mm]` returns the current time in a form such as "2007-Jun-19 04:36". The pattern following `/timestamp` specifies how to format the time and defaults to "yyyyMMddHHmm". The pattern follows conventions for the Java class `SimpleDateFormat`, where various letters are substituted with various current time elements. For more information, go to ["http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html"](http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html) for `SimpleDateFormat` class documentation.

Here are some of the supported substitutions:

y - Year, such as "2007" or "07"

M - Month, such as "April", "Apr", or "04"

w - Week in year, such as "27"

W - Week in month, such as "3"

D - Day in year, such as "194"

d - Day in month, such as "18"

E - Day in the week, such as "Tuesday" or "Tue"

a - am/pm marker, such as "PM"

H - Hour in day (0-23), such as "7"

h - Hour in am/pm (1-12), such as "11"

m - Minute in hour, such as "30"

s - Second in minute, such as "15"

S - Millisecond, such as "908"

z - Time zone such as "Pacific Standard Time", "PST", or "GMT-08:00"

Z - Time zone such as "-0800"

Repeated letters cause longer forms to be substituted. For example, "yy" substitutes the last 2 digits of the current year, whereas "yyyy" substitutes the 4-digit year number. Single quotes can be used to substitute text directly without the above interpretations.

javascript

This form executes a Javascript code fragment inside the Commander server and returns the result computed by that code.

For example, `$/javascript 4*2` returns "8.0". Javascript code can be arbitrarily long and include multiple statements. The value of the last statement is returned by the property reference.

Javascript code executes in an interpreter that provides access to Commander properties:

1. The normal way to access property values is through Javascript objects. Global Javascript objects named "server," "projects," and so on, exist and correspond to all top-level objects in an absolute property path.
For example, there is a Javascript object named `server` that corresponds to the path `/server`, and a Javascript object named `myJob` that corresponds to the path `/myJob`. You can use either `.` or `[]` notation to access properties within the object.
For a more complete list of top-level objects, see the ["Absolute property paths"](#) section above.

Examples:

```
$/javascript myJob.mightExist]
this is a safe way to refer to an optional parameter
```

```
$/javascript (myJobStep.errors > 0) ? "step failed" : "no errors"]
test a value and return another string based on the result
```

```
$/javascript server.settings.ipAddress]
refer to a property in a nested property sheet
```

```
$/javascript server["Electric Cloud"].installDirectory]
```

if the property name has a space, use `[" "]` notation

```
$[/javascript server["Electric Cloud"]["installDirectory"]]
```

do not use the `."` in front of the `["`

2. You can also call the function, `getProperty`. It takes a property name as argument and returns the value of that property. The case where this is most useful is when you want to access another special property reference.

Examples:

```
$[/javascript getProperty ("/timestamp yyyy-MMM-d hh:mm")]
```

returns the current time

```
$[/javascript getProperty ("/increment /myJob/jsCount")]
```

this is the only way to change a property via javascript

For example, consider the following property reference:

```
$[/javascript (getProperty("/myJobStep/errors") > 0) ? "step failed" : "no errors"]
```

This example returns "step failed" if the property "errors" on the current job step had a value greater than zero, and it returns "no errors" otherwise.

3. If calling the `setProperty` function, similar to `getProperty`, there are two variations on the function:
 - A *global* function variation that uses the current context object.
The global function takes 2 or 3 arguments. The 3-argument version takes a context object, a path, and a value. The 2-argument version omits the context object and uses the current context object (for example, *job step*).

Example:

```
setProperty (myProject, "foo", "bar")
```

would set the value of the "foo" property on the current project to "bar".

- An *object* function variation that can be called on objects, which uses that object as a context object. The object function takes two arguments: a "path" and a "value".

Example:

```
server.setproperty ("foo", "bar")
```

would set the "foo" property on the server object to the value "bar".

Commander Intrinsic Properties listed by object type

Each of the Commander object types [in the list below] links to a table for that object.

Each table includes a list of intrinsic properties applicable to that object and provides the property type and

description.

acl	repository
artifact	resource
artifactVersion	resourcePool
credential	resourceUsage
directoryProvider	schedule
emailConfig	server
emailNotifier	state
formalParameter	stateDefinition
gateways	step
group	transition
job	transitionDefinition
jobStep	user
logEntry	workflow
procedure	workflowDefinition
project	workspace
property	zones
propertySheet	

Commander Property Type definitions

This table provides Property Type definitions for each property listed in the following tables.
(A Property Type precedes each property description in the Description column.)

Type	Definition
boolean	One of two possible values - either <i>true</i> or <i>false</i> . These values are more frequently represented by the numbers "0" and "1", where "0" equals false and "1" equals true.
date	A millisecond precision UTC date in ISO 8601 form: [YYYY] - [MM] - [DD] T [hh] : [mm] Z For example, 2007-06-19T04:36:22.000Z
id	Each time an object is created, ElectricCommander generates a unique ID number for that object.
name	This is a unicode string value with a maximum of 255 characters.
number	This is a simple integer numeric value.
reference	This property refers to another object.
string	This is a unicode string value with a maximum CLOB size of the database, but only the first 450 characters are indexed, which means a defined search will not "see" beyond the first 450 characters.

Commander Object / Property tables

In the following tables, the Description column displays the property type preceding the property description.

Object Type: <code>acl</code> Description: An <i>acl</i> is an Access Control List.	
Property Name	Description
<code>aclId</code>	<code>id</code> : The unique identifier for this <code>acl</code> object. Other objects can refer to this <code>acl</code> by its ID.
<code>inheriting</code>	<code>boolean</code> : If true, the ACL inherits ACEs from the ACL's parent.

Object Type: *acl*

Description: An *acl* is an Access Control List.

Property Name	Description																												
ownerType	<p>This is any type of object the ACL controls and can be any of the objects listed below.</p> <table> <tr> <td>ace</td><td>project</td></tr> <tr> <td>acl</td><td>property</td></tr> <tr> <td>admin</td><td>propertySheet</td></tr> <tr> <td>artifact</td><td>repository</td></tr> <tr> <td>artifactVersion</td><td>resource</td></tr> <tr> <td>event</td><td>resourcePool</td></tr> <tr> <td>formalParameter</td><td>server</td></tr> <tr> <td>group</td><td>schedule</td></tr> <tr> <td>job</td><td>systemObject</td></tr> <tr> <td>jobStep</td><td>user</td></tr> <tr> <td>license</td><td>userSettings</td></tr> <tr> <td>notifier</td><td>workspace</td></tr> <tr> <td>procedure</td><td>plugin</td></tr> <tr> <td>procedureStep</td><td></td></tr> </table>	ace	project	acl	property	admin	propertySheet	artifact	repository	artifactVersion	resource	event	resourcePool	formalParameter	server	group	schedule	job	systemObject	jobStep	user	license	userSettings	notifier	workspace	procedure	plugin	procedureStep	
ace	project																												
acl	property																												
admin	propertySheet																												
artifact	repository																												
artifactVersion	resource																												
event	resourcePool																												
formalParameter	server																												
group	schedule																												
job	systemObject																												
jobStep	user																												
license	userSettings																												
notifier	workspace																												
procedure	plugin																												
procedureStep																													
parentId	<code>id</code> : The parent ACL.																												

[\[back to top\]](#)

Object Type: *artifact*

Description: An *artifact* is an object that contains zero or more artifact versions.

An artifact has two purposes:

1. To group artifact versions and provide a template for naming the versions
2. To restrict who can publish artifact versions, based on `groupId:artifactKey`

Property Name	Description
acl	<code>reference:acl</code>
artifactId	<code>id</code> : The artifact's ID number.

Object Type: artifact

Description: An *artifact* is an object that contains zero or more artifact versions.

An artifact has two purposes:

1. To group artifact versions and provide a template for naming the versions
2. To restrict who can publish artifact versions, based on `groupId:artifactKey`

Property Name	Description
<code>artifactKey</code>	<code>string</code> : User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>artifactName</code>	<code>name</code> : The name of this artifact.
<code>artifactVersionNameTemplate</code>	<code>name</code> : The template for artifact version names published to this artifact.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>groupId</code>	<code>id</code> : A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>

[\[back to top\]](#)

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
acl	<code>reference:</code> <code>acl</code>
artifactKey	<code>string:</code> User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName	<code>name:</code> The name of the artifact.
artifactVersionId	<code>id:</code> The Commander-generated ID number for this artifact version.
artifactVersionName	<code>name:</code> The name of the artifact version.
artifactVersionState	<code>string:</code> Possible values are: <code>available publishing unavailable</code>
buildNumber	<code>number:</code> User-defined build number component of the version attribute for the artifact version.
createTime	<code>date:</code> The time when this object was created.
description	<code>string:</code> A user-specified text description of the object.
groupId	<code>id:</code> A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
lastModifiedBy	<code>name:</code> This shows who (generally, a user name) last modified this object.
majorMinorPatch	<code>string:</code> <code>major.minor.patch</code> component of the version attribute for the artifact.
modifyTime	<code>date:</code> The time when this object was last modified.
owner	<code>name:</code> The person (user name) who created the object.

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
propertySheet	<code>reference</code> : <code>propertySheet</code>
publisherJobId	<code>id</code> : The Commander-generated ID number for the job that published the artifact version.
publisherJobName	<code>string</code> : The name of the job that published the artifact version.
publisherJobStepId	<code>id</code> : The Commander-generated ID number for the job step that published the artifact version.
qualifier	<code>string</code> : User-defined qualifier component of the version attribute for the artifact.
repositoryName	<code>name</code> : The name of the artifact repository.
version	<code>string</code> : An artifact version specification uses the following form: <i>major.minor.patch.qualifier.buildNumber</i>

[\[back to top\]](#)

Object Type: credential

Description: In Commander, a *credential* is an object that stores a username and password for later use.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
createTime	<code>date</code> : The time when this object was created.
credentialId	<code>id</code> : The credential's ID number.
credentialName	<code>name</code> : The name of this credential.

Object Type: credential

Description: In Commander, a *credential* is an object that stores a username and password for later use.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
password	<code>string</code> : The password corresponding to the user name and this credential.
projectName	<code>name</code> : The name of the <code>project</code> that contains this credential.
propertySheet	<code>reference</code> : <code>propertySheet</code>
userName	<code>name</code> : A saved string that represents the name portion of a credential, typically a user account name.

[\[back to top\]](#)

Object Type: directoryProvider

Description: A `directoryProvider` is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
commonGroupNameAttribute	<code>string</code> : 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.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
directoryProviderId	<code>id</code> : The ID of the directory provider.
domainName	<code>string</code> : The domain name from which Active Directory server(s) are automatically discovered.
emailAttribute	<code>string</code> : The attribute in a user record that contains the user's email address. If the attribute was not specified, the account name and domain name are concatenated to form an email address.
enableGroups	<code>boolean</code> : Determines whether or not external groups are enabled for the directory provider. Defaults to "true".
fullUserNameAttribute	<code>name</code> : The attribute in a user record that contains the user's full name (first and last) for display in the UI.
groupBase	<code>string</code> : This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
groupMemberAttributes	<code>string</code> : A comma-separated attribute name list that identifies a group member.
groupMemberFilter	<code>string</code> : Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	<code>name</code> : The group record that contains the name of the group.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
groupSearchFilter	string: This LDAP query is performed in the context of the groups directory to enumerate group records.
lastModifiedBy	name: This shows who (generally, a user name) last modified this object.
managerDn	name: The DN of a user who has read-only access to LDAP user and group directories.
modifyTime	date: The time when this object was last modified.
owner	name: The person (user name) who created the object.
propertySheet	reference: propertySheet
providerIndex	number: The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index "0" and one with index "1" means the providers will be searched in that numerical order.
providerName	name: This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.
providerType	string: <ldap activedirectory>
realm	string: An identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <user>@dir (where the realm is set to "dir").
url	string: The server URL is in the form <code>protocol://host:port/basedn</code> . Protocol is either ldap or ldaps (for secure LDAP).

Object Type: `directoryProvider`

Description: A `directoryProvider` is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
<code>userBase</code>	<code>string</code> : This string is prepended to the <code>basedn</code> to construct the directory DN that contains user records.
<code>userNameAttribute</code>	<code>name</code> : The attribute in a user record that contains the user's account name.
<code>userSearchFilter</code>	<code>string</code> : This LDAP query is performed in the context of the user directory to search for a user by account name. The string "{0}" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
<code>userSearchSubtree</code>	<code>boolean</code> : If true, the subtree below the user base is searched recursively.
<code>useSSL</code>	<code>boolean</code> : This flag is used to specify SSL to communicate with your Active Directory servers.

[\[back to top\]](#)

Object Type: `emailConfig`

Description: An *emailConfig* is an object that stores information created and used to communicate with the email server.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>configName</code>	<code>string</code> : The name of the email configuration.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description of the object.

Object Type: emailConfig

Description: An *emailConfig* is an object that stores information created and used to communicate with the email server.

Property Name	Description
emailConfigId	<code>id</code> : The Commander-generated ID for the email configuration.
emailConfigName	<code>string</code> : The name of the email configuration.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
mailFrom	<code>string</code> : The email address used as the email sender address for notifications.
mailHost	<code>string</code> : The name of the email server host.
mailPort	<code>number</code> : The port number for the mail server. The protocol software determines the default value (25 for SMTP and 465 for SSMTP).
mailProtocol	<code>string</code> : This is either SSMTP or SMTP (not case sensitive). Default is SMTP.
mailUser	<code>name</code> : An individual or a generic name like "Commander" -- the name of the email user on whose behalf Commander sends email notifications.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : propertySheet

[\[back to top\]](#)

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
acl	<code>reference:</code> <code>acl</code>
condition	<code>string:</code> Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
configName	<code>string:</code> The name of the email configuration.
container	<code>id:</code> The object ID to which the email notifier is attached (for example: procedure-123).
containerType	<code>string:</code> The type of object that the notifier is attached to (procedure, step, job, jobstep).
createTime	<code>date:</code> The time when this object was created.
description	<code>string:</code> A user-specified text description of the object.
destinations	<code>string:</code> A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.
emailNotifierId	<code>id:</code> The Commander-generated ID for the email notifier.
eventType	<code>string:</code> <code><onEnter onStart onCompletion></code> <code>onStart</code> triggers an email notification when the job or job step begins. <code>onCompletion</code> , default, triggers an email notification when the job finishes, no matter how it finishes.
formattingTemplate	<code>string:</code> The email address used as the email sender address for notifications.

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
notifierName	<code>name</code> : The name of the email notifier.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : propertySheet

[\[back to top\]](#)

Object Type: formalParameter

Description: A *formalParameter* is 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 type of parameters a procedure is expecting, and actual parameters provide values to use at run-time.

Property Name	Description
container	<code>string</code> : An object ID for a "container" that contains formal parameters.
containerType	<code>string</code> : The type of object containing the formal parameter.
createTime	<code>date</code> : The time when this object was created.
defaultValue	<code>string</code> : The <code>formalParameter</code> 's default value.

Object Type: `formalParameter`

Description: A *formalParameter* is 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 type of parameters a procedure is expecting, and actual parameters provide values to use at run-time.

Property Name	Description
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>expansionDeferred</code>	<code>boolean</code> : Determines whether or not the property expansion is deferred.
<code>formalParameterId</code>	<code>id</code> : This is this formal parameter's ID.
<code>formalParameterName</code>	<code>name</code> : This is this formal parameter's name.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>required</code>	<code>boolean</code> : If true, a value for this parameter must be supplied when the procedure is called.
<code>type</code>	<code>name</code> : The custom type of <code>formalParameter</code> .

[\[back to top\]](#)

Object Type: gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
acl	<code>reference</code> : acl
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
gatewayDisabled	<code>boolean</code> : If set to 1 (true), the gateway is disabled.
gatewayId	<code>id</code> : The unique Commander-generated ID for this gateway.
gatewayName	<code>string</code> : The name of the gateway.
hostName1	<code>string</code> : The agent host name where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option if you want to use the host name from <i>Resource1</i> 's definition.
hostName2	<code>string</code> : The agent host name where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option if you want to use the host name from <i>Resource2</i> 's definition.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
port1	<code>number</code> : The port number used by <i>Resource1</i> - defaults to the port number used by the resource.

Object Type: gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
port2	<code>number</code> : The port number used by <i>Resource2</i> - defaults to the port number used by the resource.
propertySheet	<code>reference</code> : <code>propertySheet</code>
resourceName1	<code>string</code> : The name of your choice for the first of two required gateway resources. Do not include "spaces" in a resource name.
resourceName2	<code>string</code> : The name of your choice for the second of two required gateway resources. Do not include "spaces" in a resource name.

[\[back to top\]](#)

Object Type: group

Description: This is any *group* of users known to the Commander server, including groups defined locally within the server as well as those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
createTime	<code>date</code> : The time when this object was created.
groupId	<code>id</code> : The unique Commander-generated group ID.
groupName	<code>name</code> : This is this group's name.

Object Type: group

Description: This is any *group* of users known to the Commander server, including groups defined locally within the server as well as those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>mutable</code>	<code>boolean</code> : If true, the member list of this group is editable within ElectricCommander via the web UI or the <code>modifyGroup</code> API.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>providerName</code>	<code>name</code> : The name of the <code>directory provider</code> that controls this group.

[\[back to top\]](#)**Object Type: job**

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>abortStatus</code>	<code>string</code> : If set, indicates the step was aborted. Values will be either <code>ABORT</code> or <code>FORCE_ABORT</code> - indicating how the step was aborted.
<code>abortedBy</code>	<code>name</code> : This is the user who issued the "abort."

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>actualParameters</code>	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>callingState</code>	<code>string</code> : The full property path to the "calling state", which can appear on <code>subjobs</code> and <code>subworkflows</code> of a workflow.
<code>callingStateId</code>	<code>id</code> : The Commander-generated ID number for the calling state.
<code>combinedStatus</code>	Provides more inclusive step status output - the resulting query output may contain up to three sub-elements: <code>status message properties</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>credentialName</code>	<code>name</code> : The name of the <code>credential</code> being used for impersonation when the job runs commands on a resource.
<code>deleted</code>	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
<code>directoryName</code>	<code>name</code> : The name of this job's directory within each workspace for this job.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>elapsedTime</code>	<code>number</code> : The number of milliseconds between the start and end times for the job.
<code>errorCode</code>	<code>errorCode</code> : When the outcome is <code>error</code> , this property displays the error code, identifying which error occurred.
<code>errorMessage</code>	<code>string</code> : When the outcome is <code>error</code> , this property displays the error description.
<code>external</code>	<code>boolean</code> : If "true", the job is external.
<code>finish</code>	<code>date</code> : The time this job completed.
<code>jobId</code>	<code>id</code> : This is this job's ID number, which is a UUID.
<code>jobName</code>	<code>name</code> : This is this job's name.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>launchedByUser</code>	<code>name</code> : 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.
<code>licenseWaitTime</code>	The sum of time all job steps had to wait for a license.
<code>liveProcedure</code>	<code>name</code> : The current procedure name from which this job was created – if the procedure was renamed since the job launched, this will be the procedure's new name, and if the procedure was deleted, this will be null.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>liveSchedule</code>	<code>name</code> : The current schedule name that launched this job – if the schedule was renamed since the job was launched, this will be the schedule's new name, and if the schedule was deleted, this will be null.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>outcome</code>	The overall result of the job: <code>success</code> , <code>warning</code> , <code>error</code> , or <code>skipped</code> .
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>priority</code>	Values can be <code>low</code> , <code>normal</code> (default), <code>high</code> , or <code>highest</code> .
<code>procedureName</code>	<code>name</code> : The name of the <code>procedure</code> that defines the job's steps.
<code>projectName</code>	<code>name</code> : The name of the <code>project</code> that contains this job.
<code>propertySheetId</code>	<code>reference</code> : <code>propertySheet</code>
<code>resourceWaitTime</code>	The sum of time all job steps had to wait for a resource. 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.
<code>runAsUser</code>	<code>name</code> : The name of the user being impersonated in this job.
<code>scheduleName</code>	<code>name</code> : The schedule name that launched this job – this field differs from <code>liveSchedule</code> in that it is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
<code>start</code>	<code>date</code> : The time when this job began executing.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>status</code>	Possible values are: <code>pending</code> - the job was created, but it is waiting for prerequisite steps to complete <code>runnable</code> - the job step is waiting for a resource <code>scheduled</code> - the job was assigned a resource, but the command has not started running <code>running</code> - the job/job step is running a command on the assigned resource <code>completed</code> - the job/job step has completed
<code>totalWaitTime</code>	The total sum of license, resource, a precondition, and workspace wait times job steps were restricted and had to wait to process.
<code>workspaceWaitTime</code>	The sum of time all job steps had to wait for a workspace.

[\[back to top\]](#)

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>abortStatus</code>	<code>string</code> : If set, indicates the step was aborted. Values will be either <code>ABORT</code> or <code>FORCE_ABORT</code> - indicating how the step was aborted.
<code>abortedBy</code>	<code>name</code> : The user who issued the "abort."

Object Type: <code>jobStep</code> Description: A <i>jobStep</i> is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a job .	
Property Name	Description
<code>acl</code>	reference: <code>acl</code>
<code>actualParameters</code>	reference: <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>alwaysRun</code>	boolean: If true, this step runs even if the job is aborted before the step completes. Note that a "force abort" will abort an <code>alwaysRun</code> step.
<code>assignedResourceName</code>	name: The name of the resource assigned to this step by the step scheduler.
<code>broadcast</code>	boolean: If true, this step runs on all resources in a pool.
<code>combinedStatus</code>	Provides more inclusive step status output - the resulting query output may contain up to three sub-elements: <code>status message properties</code>
<code>command</code>	string: This property specifies the command this step runs.
<code>condition</code>	string: If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
<code>conditionExpanded</code>	boolean: The result of the expansion on the step condition.
<code>createTime</code>	date: The time when this object was created.

Object Type: <code>jobStep</code> Description: A <i>jobStep</i> is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a job .	
Property Name	Description
<code>delayUntil</code>	For a step that was rescheduled due to a resource or workspace problem, this is the next time the step will be eligible to run.
<code>elapsedTime</code>	number : The number of milliseconds between the start and end times for the <code>jobStep</code> .
<code>errorCode</code>	errorCode : This property appears when the outcome is <code>error</code> and displays the error code, identifying which error occurred.
<code>errorHandling</code>	<p>This is the error handling policy copied from the procedure step and indicates how the step responds to errors:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</p> <p><code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure.</p> <p><code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run.</p> <p><code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps.</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>ignore</code> - Continues as if the step succeeded.</p>
<code>errorMessage</code>	string : When the outcome is <code>error</code> , this property displays an error message description.
<code>exclusive</code>	boolean : If true, this step acquires and retains its resource exclusively.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>exclusiveMode</code>	<code>string</code> : Possible values are: none, job, step, call
<code>exitCode</code>	<code>number</code> : This is this step's exit code.
<code>external</code>	<code>boolean</code> : If "true", the job is external.
<code>finish</code>	<code>date</code> : The time when this job step completed.
<code>hostName</code>	<code>name</code> : The name of the host where this step was invoked (copied from the resource)
<code>job</code>	<code>id</code> : This is the ID number of the job that owns the job step. The string representation of the job is its name, so <code>\$(job)</code> evaluates to the job name, but <code>\$(job/foo)</code> is also legal and refers to the foo property of the job.
<code>jobId</code>	<code>id</code> : This is this job's ID number, which is a UUID.
<code>jobName</code>	<code>name</code> : This is the name of this step's job.
<code>jobStepId</code>	<code>id</code> : This is this step's ID number.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>licenseWaitTime</code>	The length of time this job step had to wait to process because the license limit was reached or exceeded.
<code>liveProcedure</code>	<code>string</code> : 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.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>liveProcedureStep</code>	name: This property shows the current procedure name for the procedure step from which this job step was created – if the procedure step was renamed since the job was launched, this will be the procedure step's new name, and if the procedure step was deleted, this will be null.
<code>logFileName</code>	name: The name of the log file produced by this step, relative to the job's workspace directory.
<code>modifyTime</code>	date: The time when this object was last modified.
<code>outcome</code>	string: The overall result of the job - possible values are: <code>success</code> , <code>warning</code> , <code>error</code> , or <code>skipped</code>
<code>owner</code>	name: The person (user name) who created the object.
<code>parallel</code>	boolean: If true, this step runs in parallel with other adjacent steps also marked to run in parallel.
<code>postExitCode</code>	number: This is this step's post processor exit code.
<code>postLogFileName</code>	name: The log file name produced by this step's post processor.
<code>postProcessor</code>	string: The post processor name used to gather information about this step.
<code>precondition</code>	string: By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated. A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.

Object Type: <code>jobStep</code> Description: A <i>jobStep</i> is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a job .	
Property Name	Description
<code>procedureName</code>	<code>name</code> : The name of the procedure that contains this <code>jobStep</code> .
<code>projectName</code>	<code>name</code> : The name of the project that contains this <code>jobStep</code> .
<code>propertySheet</code>	<code>reference</code> : propertySheet
<code>releaseExclusive</code>	<code>boolean</code> : A Boolean value indicating whether this step should release its resource upon completion.
<code>releaseMode</code>	<code>string</code> : Possible values are: none, release, releasetojob
<code>resourceName</code>	<code>name</code> : The name of the resource or pool this step should use to run on.
<code>resourceSource</code>	This property indicates whether the resource for the job step is explicitly specified by the step or was defaulted from the procedure: procedure or <code>procedureStep</code> .
<code>resourceWaitTime</code>	The length of time this job step stalled because it could not get a resource. 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.
<code>retries</code>	<code>number</code> : This is a number - the number of retries before the request times out.
<code>runAsUser</code>	<code>name</code> : The name of the user being impersonated in this job step.
<code>runnable</code>	<code>date</code> : The time when the step became runnable.
<code>runTime</code>	<code>number</code> : The number of milliseconds the step command spent running on a resource.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>shell</code>	name : The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a " {0} " marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
<code>start</code>	date : The time when this job step began executing.
<code>status</code>	Possible values are: <code>pending</code> - the job step was created, but it is waiting for prerequisite steps to complete <code>runnable</code> - the job step is waiting for a resource <code>scheduled</code> - the job step was assigned a resource, but the command has not started running <code>running</code> - the job/job step is running a command on the assigned resource <code>completed</code> - the job/job step has completed
<code>stepName</code>	name : This is this step's name.
<code>subprocedure</code>	name : The nested procedure name to call when this step executes.
<code>subproject</code>	name : This property appears if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)
<code>timeLimit</code>	number : The maximum length of time this step is allowed to run.
<code>timeout</code>	date : The time when this job step will be automatically aborted if it has not yet completed.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>totalWaitTime</code>	The sum of resource, workspace, and license wait times for this job step.
<code>waitTime</code>	number : The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).
<code>workingDirectory</code>	name : The name of this step's working directory.
<code>workspaceName</code>	name : The workspace name used by the <code>jobStep</code> object.
<code>workspaceWaitTime</code>	The time this job step had to wait because no workspace was found or available.

[\[back to top\]](#)

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the Commander system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The Commander server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
category	(currently not used)
container	<code>string</code> : Typically, this is the type and name of the workflow or job with a corresponding ID.
containerName	<code>name</code> : The name of the container.
containerType	<code>string</code> : The type of object the log entry pertains to (for example, procedure, job step, step).
deleted	<code>byte</code> : The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (which means "deleted" is not set).
logEntryId	<code>id</code> : The log entry Commander-generated ID.
message	<code>string</code> : The message text will either be informational or display the warning or error message. The message may contain important information about a resource or workspace issue.
principal	<code>string</code> : The user or project principal from the session that was active when the event occurred.

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the Commander system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The Commander server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
severity	<code>string</code> : Severity can be either TRACE, DEBUG, INFO, WARN, ERROR
subject	<code>string</code> : The object associated with the message.
subjectName	<code>name</code> : The name of the object associated with the message.
subjectType	<code>string</code> : (similar to <code>container</code>) Refers to the object the event concerns. This may be the same as the container, or it may be a different object that is related to the event in some manner.
time	<code>string</code> : The time the event was logged.

[\[back to top\]](#)

Object Type: procedure

Description: A *procedure* describes a series of actions to be performed on one or more resources. A procedure contains steps and properties and can define formal parameters.

Property Name	Description
acl	reference: acl
createTime	date : The time when this object was created.
credentialName	name : The name of the credential assigned to this job step.
description	string : A user-specified text description of the object.
jobNameTemplate	name : The template used to name jobs created from this procedure.
lastModifiedBy	name : This shows who (generally, a user name) last modified this object.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
procedureId	id : This is this procedure's ID number.
procedureName	name : This is this procedure's name.
projectName	name : The name of the project that contains this procedure.
propertySheet	reference: propertySheet
resourceName	name : The name of the resource or pool this procedure should use to run on.
workspaceName	name : The workspace name used by the procedure object.

[\[back to top\]](#)

Object Type: project

Description: A *project* is an object used in Commander to organize information. A project contains procedures, schedules, credentials, and properties.

Property Name	Description
acl	<code>reference: acl</code>
createTime	<code>date</code> : The time when this object was created.
credentialName	<code>name</code> : The name of the credential assigned to this project.
deleted	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
pluginName	<code>name</code> : The name of the plugin associated with this project.
projectId	<code>id</code> : This is this project's ID.
projectName	<code>name</code> : The name of the project.
propertySheet	<code>reference: propertySheet</code>
resourceName	<code>name</code> : The name of the resource.
workspaceName	<code>name</code> : This is the workspace name used by the project.

[\[back to top\]](#)

Object Type: property

Description: A *property* is a string value with a name. Properties can have arbitrary names and values. You can attach properties to any object in the Commander system, such as a project, procedure, or job. After a property is created, it is stored in the Commander database. You can retrieve and/or modify the value later, and you can delete properties you no longer need. Properties provide a flexible and powerful mechanism to manage data about your builds.

Note: The name "properties" is NOT a valid property name.

Property Name	Description
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
expandable	<code>boolean</code> : If set to true, the property value will undergo string expansion when it is retrieved.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
canonicalPath	<code>string</code> : The path that specifies the property object.
propertyId	<code>id</code> : This property's ID number.
propertyName	<code>name</code> : This property's name.
propertySheetId	<code>reference</code> : <code>propertySheet</code> – If the property is a nested property sheet, the <code>propertySheet</code> refers to the nested sheet.
value	<code>string</code> : The property or actual parameter's value - if this property is a string property.

[\[back to top\]](#)

Object Type: `propertySheet`

Description: A property value can be a simple string or a nested *propertySheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.

Most objects have an associated property sheet that contains "custom properties" created by user scripts.

Property Name	Description
<code>acl</code>	<code>reference: acl</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheetId</code>	<code>id</code> : This property sheet's ID.

[\[back to top\]](#)

Object Type: `repository`

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on 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.

Property Name	Description
<code>acl</code>	<code>reference: acl</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description for this object.

Object Type: repository

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on 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.

Property Name	Description
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : <code>propertySheet</code>
repositoryDisabled	<code>boolean</code> : Determines whether the repository is disabled. Default is "false".
repositoryId	<code>id</code> : The Commander-generated ID number of the repository.
repositoryIndex	<code>number</code> : The order of the repository, within a list of repositories.
repositoryName	<code>name</code> : The name of the repository.
url	<code>string</code> : The server URL is in the form <code>protocol://host:port/</code> . Typically, the repository server is configured to listen on port 8200 for <code>https</code> requests, so a typical URL looks like <code>https://host:8200/</code> .
zoneName	<code>name</code> : The name of the zone where this repository is or will reside.

[\[back to top\]](#)

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
acl	<code>reference: acl</code>
agentState	<code>string</code> : Specific information about an agent, including the state of the agent. Possible values are: <code>unknown alive down</code>
artifactCacheDirectory	<code>string</code> : The directory on the agent host where retrieved artifacts are stored.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
exclusiveJobId	<code>id</code> : The ID number of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobName	<code>name</code> : The name of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobStepId	<code>id</code> : The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
exclusiveJobStepName	<code>name</code> : 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.
gateways	The gateway name(s) associated with this resource.

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
hostName	<code>name</code> : The computer name or IP address for the machine containing the Commander agent for this resource.
hostOS	<code>string</code> : The full name of the host operating system, plus its version. However, if this host is a proxy, "proxied" appears as the host description/name.
hostPlatform	<code>string</code> : Examples for "platform" are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, "proxied" appears as the hostPlatform name.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
lastRunTime	<code>date</code> : The most recent time a job step ran on the resource.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
pools	<code>string</code> : 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	<code>number</code> : The port number for this resource.
propertySheet	<code>reference</code> : <code>propertySheet</code>
proxyCustomization	<code>string</code> : This property displays the customization of the Commander proxy agent.

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
proxyHostName	<code>name</code> : The name of the Commander agent being used to proxy to another agent, the proxy target.
proxyPort	<code>number</code> : The port number of the Commander agent being used to proxy to another agent, the proxy target.
proxyProtocol	<code>name</code> : The name of the proxy agent protocol used for communication from the Commander proxy agent to the proxy target - default is SSH.
repositoryNames	A "new line" separated list of repository names.
resourceDisabled	<code>boolean</code> : A Boolean value indicating whether this resource was disabled.
resourceId	<code>id</code> : This is this resource's ID.
resourceName	<code>name</code> : The name of the resource or pool.
shell	<code>name</code> : The shell used to execute the step's commands on a resource. If no shell was defined on a step, the shell defined on the resource is used, or if no shell was defined on the resource, a default shell is used. For shells: The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
stepCount	<code>number</code> : The current number of executing steps on this resource.
stepLimit	<code>number</code> : This property specifies the maximum number of steps that can run on this resource at one time.

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server though a Commander agent proxy machine.

Property Name	Description
trusted	<code>boolean</code> : If "true", this agent is trusted.
useSSL	<code>boolean</code> : A Boolean value indicating whether this resource uses SSL for communication.
workspaceName	<code>name</code> : The workspace name used by this resource.
zoneName	<code>string</code> : The name of the zone where this resource resides.

[\[back to top\]](#)**Object Type: resourcePool**

Description: A *resource pool* is a container for a group of resources.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
autoDelete	<code>boolean</code> : If "true", the resource pool is deleted when the last resource is removed or deleted.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
lastResourceUsed	<code>name</code> : The name of the most recently used resource from the pool.

Object Type: resourcePool

Description: A *resource pool* is a container for a group of resources.

Property Name	Description
modifyTime	<code>date</code> : The time when this object was last modified.
orderingFilter	<code>string</code> : 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.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : propertySheet
resourcePoolDisabled	<code>boolean</code> : A Boolean value indicating whether this resource pool was disabled.
resourcePoolId	<code>id</code> : This resource pool's ID number.
resourcePoolName	<code>name</code> : The name of the resource pool.

[\[back to top\]](#)

Object Type: resourceUsage

Description: Provides usage information for all resources in the system.

For any job step running on a resource, there is a resource usage record that contains the ID and name of the job, job step, and resource.

Property Name	Description
jobId	<code>id</code> : This is this job's ID number, which is a UUID.
jobName	<code>name</code> : The name of the job.
jobStepId	<code>id</code> : The ID number of the job step.
jobStepName	<code>name</code> : The name of the job step.

Object Type: resourceUsage

Description: Provides usage information for all resources in the system.

For any job step running on a resource, there is a resource usage record that contains the ID and name of the job, job step, and resource.

Property Name	Description
licenseWaitTime	The time this job step had to wait because no license was found or available.
resourceId	<code>id</code> : The Commander-generated resource ID number.
resourceName	<code>name</code> : The name (or names) of the resource.
resourcePoolId	<code>id</code> : The Commander-generated resource pool's ID number.
resourcePoolName	<code>name</code> : The name of the resource pool.
resourceUsageId	<code>id</code> : The unique ID of the resource usage record.
resourceWaitTime	The time this job step had to wait because no resource was found or available. 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.
waitReason	Possible values are: <code>license</code> , <code>resource</code> , or <code>workspace</code> .
workspaceWaitTime	The time this job step had to wait because no workspace was found or available.

[\[back to top\]](#)

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>

Object Type: <code>schedule</code> Description: <i>Schedules</i> are used to execute procedures and determine when specific procedures run.	
Property Name	Description
<code>actualParameters</code>	reference: <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>beginDate</code>	string: The first day on which the schedule is active, in the form YYYY-MM-dd. For example, "2009-06-25"
<code>createTime</code>	date: The time when this object was created.
<code>credentialName</code>	name: The name of the <code>credential</code> being used for impersonation when the schedule launches a job.
<code>description</code>	string: A user-specified text description of the object.
<code>endDate</code>	string: The end of the range of dates the schedule is active, in the form: YYYY-MM-dd. The actual end date is not included in the range. For example, "2009-06-25"
<code>interval</code>	string: A floating point number that represents the time to wait between invocations of the schedule.
<code>intervalUnits</code>	string: These are the units to use when interpreting the <i>interval</i> value. The <i>units</i> can be <code>hours</code> , <code>minutes</code> , <code>seconds</code> , or <code>continuous</code> .
<code>lastModifiedBy</code>	name: This shows who (generally, a user name) last modified this object.
<code>lastRunTime</code>	date: The last time a job was launched by a schedule.

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
misfirePolicy	string: The policy the schedule uses when it is unable to launch a job at the scheduled time: ignore - wait until the next scheduled time to run runOnce - run immediately, then resume normal scheduling
modifyTime	date: The time when this object was last modified.
monthDays	string: This property specifies which days of the month this schedule should run - shown as a space-separated list of numbers (1-31).
owner	name: The person (user name) who created the object.
priority	string: Values can be low, normal (default), high, or highest
procedureName	name: The name of the procedure that contains this schedule.
projectName	name: The name of the project that contains this schedule.
propertySheet	reference: propertySheet
scheduleDisabled	boolean: A Boolean value indicating whether this schedule was disabled.
scheduleId	id: This schedule's ID number.
scheduleName	name: This property displays this schedule's name and differs from the liveSchedule property found under a job in that the liveSchedule property is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
startTime	string: The time of day when this schedule should start running, in the form: HH:mm:ss

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
stopTime	string: The time of day when this schedule should stop running, in the form: HH:mm:ss
timeZone	string: A Java-compatible time zone string.
weekDays	string: This property specifies which days of the week this schedule should run. This is a space-separated list of MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY SUNDAY. (Note: These are not localized strings.)

[\[back to top\]](#)

Object Type: server

Description: This is the Commander server. There is only one such object in your database.

Read-only server properties

Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricCommander server location.

/server/hostname - returns the Commander server name

/server/hostIP - returns the Commander server IP address

These properties are especially useful for building a URL link to an ElectricCommander web page. For example, a link in an email notifier that links back to a Job Details page.

A sample link: `https://$[/server/hostname]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
acl	reference: acl
createTime	date: The time when this object was created.
hostIP	string: The server's IP address.

Object Type: `server`

Description: This is the Commander server. There is only one such object in your database.

Read-only server properties

Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricCommander server location.

`/server/hostname` - returns the Commander server name

`/server/hostIP` - returns the Commander server IP address

These properties are especially useful for building a URL link to an ElectricCommander web page. For example, a link in an email notifier that links back to a Job Details page.

A sample link: `https://$[/server/hostname]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
<code>hostname</code>	<code>name</code> : Generally refers to the computer name or IP address for the machine containing the Commander server.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>serverId</code>	<code>id</code> : The server's ID number.
<code>serverName</code>	<code>string</code> : The Commander server's name - defaults to "server".

[\[back to top\]](#)

Object Type: state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (from the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
active	<code>boolean</code> : If "true", the state of the workflow is active.
actualParameters	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
errorMessage	<code>string</code> : When the outcome is <code>error</code> , this property displays an error message description.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the <code>project</code> that contains this state.
propertySheet	<code>reference</code> : <code>propertySheet</code>
stateId	<code>id</code> : The Commander-generated ID number for the state.

Object Type: state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (from the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
stateName	name : The name of the state.
subjob	name : The name of the subjob.
subparameters	reference : propertySheet
subprocedure	name : The name of the nested procedure called when a step runs.
subproject	string : If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	name : Name of the starting state for the workflow launched when the state is entered.
subworkflow	name : The name of the subworkflow - a workflow called by another workflow.
subworkflowDefinition	name : The name of the subworkflow definition.
workflow	name : The name of the workflow.

[\[back to top\]](#)

Object Type: stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
acl	<code>reference: acl</code>
actualParameters	<code>reference: propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the project that contains this state definition.
propertySheet	<code>reference: propertySheet</code>
startable	<code>boolean</code> : "True" means this state definition can be the initial state of an instantiated workflow.
stateDefinitionId	<code>id</code> : The Commander-generated ID number for the state definition.
stateDefinitionName	<code>name</code> : The name of the state definition.

Object Type: stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
subprocedure	<code>name</code> : The name of the nested procedure called when a step runs.
subproject	<code>string</code> : If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	<code>name</code> : Name of the starting state for the workflow launched when the state is entered.
subworkflowDefinition	<code>name</code> : The name of the subworkflow definition.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.

[\[back to top\]](#)

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
actualParameters	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
alwaysRun	<code>boolean</code> : If true, this step runs even if the job is aborted before the step completes. Note that a force abort will abort an <code>alwaysRun</code> step.
broadcast	<code>boolean</code> : If true, this step will run on all resources in a pool.
command	<code>string</code> : This property specifies the command this step runs.
condition	<code>string</code> : If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
createTime	<code>date</code> : The time when this object was created.
credentialName	<code>name</code> : The credential name assigned to this step.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
errorHandling	<p>This is the error handling policy copied from the procedure step and indicates how the step responds to errors:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</p> <p><code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure.</p> <p><code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run.</p> <p><code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps.</p> <p><code>ignore</code> - Continues as if the step succeeded.</p>
exclusive	<code>boolean</code> : If true, this step acquires and retains its resource exclusively.
exclusiveMode	<code>string</code> : Possible values are: none, job, step, call
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
logFileName	<code>name</code> : The name of the log file produced by this step, relative to the job's workspace directory.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
parallel	<code>boolean</code> : If true, this step runs in parallel with other adjacent steps also marked to run in parallel.
postLogFileName	<code>name</code> : This property displays the log file name produced by this step's post processor.
postProcessor	<code>string</code> : This property displays the post processor name that is used to gather information about this step. Typically, this is either <code>postp</code> or <code>postp <options></code> , or an appropriate command or path including options to a postprocessor available on the Commander server.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
precondition	<p>string: By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p>
procedureName	name: The name of the procedure that contains this <i>step</i> .
projectName	name: The name of the project that contains this step.
propertySheet	reference: <code>propertySheet</code>
releaseExclusive	boolean: A Boolean value indicating whether this step should release its resource upon completion.
releaseMode	string: Possible values are: none, release, releasetojob
resourceName	name: The resource's name this step should use to run on.

Object Type: `step`

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
<code>shell</code>	<code>name</code> : The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
<code>stepId</code>	<code>id</code> : The step's ID number.
<code>stepName</code>	<code>name</code> : The step's name.
<code>subprocedure</code>	<code>name</code> : The nested procedure name to call when this step executes.
<code>subproject</code>	<code>name</code> : This property is displayed if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)
<code>timeLimit</code>	<code>number</code> : A floating point number that specifies the maximum length of time this step is allowed to run.
<code>timeLimitUnits</code>	The units to use when interpreting the time limit. Units can be <code>hours</code> , <code>minutes</code> , or <code>seconds</code> .
<code>workingDirectory</code>	<code>name</code> : The name of the step's working directory.
<code>workspaceName</code>	<code>name</code> : The workspace name used by this step.

[\[back to top\]](#)

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
<code>acl</code>	<code>reference: acl</code>
<code>actualParameters</code>	<code>reference: propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
<code>condition</code>	<code>string</code> : If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>index</code>	The numeric index of a transition that indicates the transition order in the state definition's transition list.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>projectName</code>	<code>name</code> : The name of the <code>project</code> that contains this transition.
<code>propertySheet</code>	<code>reference: propertySheet</code>
<code>stateName</code>	<code>name</code> : The name of the state.
<code>targetState</code>	<code>string</code> : The target state for the transition definition.
<code>transitionId</code>	<code>id</code> : The Commander-generated ID for the transition.

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
transitionName	<code>name</code> : The name of the transition.
trigger	<code>string</code> : Possible values are: <code>onEnter</code> - before any actions <code>onStart</code> - after a subjob or workflow is created <code>onCompletion</code> - after a subjob or workflow completes <code>manual</code> - when a user manually requests a transition
workflowName	<code>name</code> : The name of the workflow.

[\[back to top\]](#)

Object Type: transitionDefinition

Description: A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`)).

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
actualParameters	<code>reference</code> : <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter passed to the target state.
condition	<code>string</code> : If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.
createTime	<code>date</code> : The time when this object was created.

Object Type: transitionDefinition

Description: A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (onEnter|onStart|onCompletion|manual)).

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
index	The numeric index of a transition that indicates the transition order in the state definition's transition list.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the project that contains this transition definition.
propertySheet	<code>reference</code> : propertySheet
stateDefinitionName	<code>name</code> : The name of the state definition.
targetState	<code>string</code> : The target state for the transition definition.
transitionDefinitionId	<code>id</code> : The Commander-generated ID for the transition definition.
transitionDefinitionName	<code>name</code> : The name of the transition definition.
trigger	<code>string</code> : Possible values are: onEnter - before any actions onStart - after a subjob or workflow is created onCompletion - after a subjob or workflow completes manual - when a user manually requests a transition
workflowDefinitionName	<code>name</code> : The name of the workflow definition.

[\[back to top\]](#)

Object Type: <code>USER</code>	
Description: <i>User</i> refers to any user currently logged into the Commander system or any user who may use Commander, but may not be currently logged in.	
Property Name	Description
<code>acl</code>	<code>reference:</code> <code>acl</code>
<code>createTime</code>	<code>date:</code> The time when this object was created.
<code>email</code>	<code>name:</code> Displays this user's email address.
<code>fullUserName</code>	<code>name:</code> The user's real name.
<code>lastModifiedBy</code>	<code>name:</code> This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date:</code> The time when this object was last modified.
<code>mutable</code>	<code>boolean:</code> If true, the user is editable within ElectricCommander via the web UI or the <code>modifyUser</code> API.
<code>owner</code>	<code>name:</code> The person (user name) who created the object.
<code>propertySheet</code>	<code>reference:</code> <code>propertySheet</code>
<code>providerName</code>	<code>name:</code> The name of the <code>directory provider</code> that controls this user.
<code>userId</code>	<code>id:</code> A unique ID number for a user object.
<code>userName</code>	<code>name:</code> Displays this user's name and also appears in a group object and displays the name of a user who belongs to this group.

[\[back to top\]](#)

Object Type: workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting state, parameters to the initial state, "complete" - a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
activeState	<code>string</code> : The name of the active state on the workflow object.
actualParameters	<code>reference</code> : <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
callingState	<code>string</code> : The full property path to the state that created this workflow.
callingStateId	<code>id</code> : The ID number for the full property path to the state that created this workflow.
completed	<code>boolean</code> : If "true", the workflow is completed and no additional transactions will be evaluated.
createTime	<code>date</code> : The time when this object was created.
deleted	<code>boolean</code> : The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
elapsedTime	The time this object ran from start to finish.
finish	<code>date</code> : The date/time when this object finished.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
launchedByUser	<code>string</code> : The name of the user or project principal that explicitly launched the workflow.
liveWorkflowDefinition	<code>name</code> : The current workflow definition name for the workflow definition from which the workflow was created.

Object Type: workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting state, parameters to the initial state, "complete" - a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the project that contains this workflow.
propertySheet	<code>reference</code> : propertySheet
start	<code>date</code> : The date/time when this workflow began to run.
startingState	<code>string</code> : The initial state of the workflow.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.
workflowId	<code>id</code> : The Commander-generated ID for the workflow.
workflowName	<code>name</code> : The name of the workflow.

[\[back to top\]](#)

Object Type: workflowDefinition

Description: A *workflowDefinition* object contains state and transition definitions, including the workflow name template (analogous to a job name template on a procedure), and ordered sets of state and transition definitions.

Property Name	Description
acl	<code>reference</code> : acl
createTime	<code>date</code> : The time when this object was created.

Object Type: workflowDefinition

Description: A *workflowDefinition* object contains state and transition definitions, including the workflow name template (analogous to a job name template on a procedure), and ordered sets of state and transition definitions.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the project that contains this workflow definition.
propertySheet	<code>reference</code> : propertySheet
workflowDefinitionId	<code>id</code> : The Commander-generated ID for the workflow definition.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.
workflowNameTemplate	<code>string</code> : Template used to determine the default names for workflows launched from a workflow definition.

[\[back to top\]](#)

Object Type: workspace

Description: ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and Commander automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
acl	reference : acl
agentDrivePath	name : The drive-letter based mount point for this workspace on Windows agents.
agentUncPath	name : The UNC path for this workspace on Windows agents.
agentUnixPath	name : The UNIX path for this workspace on UNIX agents.
createTime	date : The time when this object was created.
credentialName	name : The name of the credential being used when a resource connects to a network share while setting up the workspace.
description	string : A user-specified text description of the object.
lastModifiedBy	name : This shows who (generally, a user name) last modified this object.
local	boolean : If "true", this workspace is local.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
propertySheet	reference : propertySheet

Object Type: workspace

Description: ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and Commander automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
<code>workspaceDisabled</code>	<code>boolean</code> : A Boolean value that indicates whether this workspace was disabled.
<code>workspaceId</code>	<code>id</code> : The workspace's ID number.
<code>workspaceName</code>	<code>name</code> : The workspace's name.
<code>zoneName</code>	<code>name</code> : The name of the zone where this workspace resides.

[\[back to top\]](#)

Object Type: Zone

Description: Commander provides the ability to create zones—often used to enhance security.

- 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—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the [Gateways](#) Help topic.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>

Object Type: Zone

Description: Commander provides the ability to create zones—often used to enhance security.

- 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—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the [Gateways](#) Help topic.

Property Name	Description
createTime	date : The time when this object was created.
description	string : A user-specified text description for the object.
lastModifiedBy	name : This person (generally, a user name) last modified this object.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
propertySheet	reference : propertySheet
resources	string : A space-separated list of resources contained in this zone.
zoneId	id : The Commander-generated ID for this zone.
zoneName	string : The name of the zone.

Property error codes

The following list provides error codes that may appear as a value within an `errorCode` property.

ABORTED

ACCESS_DENIED

AGENT_BAD_WORKINGDIR

AGENT_FAILED_CONNECT_BROKER

AGENT_FAILED_CREATE_FILE
AGENT_FAILED_CREATE_WORKSPACE
AGENT_FAILED_IMPERSONATION
AGENT_FAILED_MAP_DRIVE
AGENT_FAILED_PROXYTARGET_PING
AGENT_INCOMPATIBLE_VERSION
AGENT_INTERNAL_ERROR
AGENT_INVALID_CWD
AGENT_INVALID_MESSAGE
AGENT_INVALID_PING_TOKEN
AGENT_INVALID_WORKSPACE
AGENT_IO_CONNECTION_REFUSED
AGENT_IO_CONNECTION_RESET
AGENT_IO_ERROR
AGENT_IO_NO_ROUTE_TO_HOST
AGENT_IO_PORT_UNREACHABLE
AGENT_MALFORMED_XML
AGENT_NONABSOLUTE_PATH
AGENT_NONEXISTENT_DIR
AGENT_NONEXISTENT_FILE
AGENT_PING_FAILED
AGENT_STREAM_STOPPED
AGENT_TIMEOUT
AGENT_UNKNOWN_CMDID
AGENT_UNKNOWN_COMMAND
AGENT_WRONG_FILE_TYPE
BAD_AGENT_RESPONSE
BAD_LOGFILE_PATH
CANCELED
CIRCULAR_PROCEDURE_REFERENCE
CORRUPT_CREDENTIAL

DUPLICATE_JOB_NAME
EMPTY_SUBPROCEDURE
FAILED_JOB_RENAME
FORMAL_PARAMETER_ERROR
INTERNAL_ERROR
INVALID_EMAIL_HEADER
MY_EVENT_EXPANSION_ERROR
NO_EMAIL_HEADERS_SEPARATOR
NONEXISTENT_EMAIL_CONFIG
NONEXISTENT_PROCEDURE
NONEXISTENT_RESOURCE
NONEXISTENT_WORKSPACE
NOTIFIER_EXPANSION_ERROR
POST_PROCESSOR_ERROR
POST_PROCESSOR_OUTPUT_FAILURE
PROPERTY_REFERENCE_ERROR
RESOURCE_WITHOUT_HOSTNAME
SERVER_SHUTDOWN
TIMEOUT
UNKNOWN_HOST

Postprocessors: Collecting Data for Reports

[Postp](#)

[Extending postp: matchers](#)

[Postp functions](#)

[Integration with the Commander user interface](#)

[Postp integration with Java Tools](#)

Overview

In ElectricCommander, reporting is divided into two phases.

- The first phase is data collection: interesting information is extracted from job step logs and saved in the ElectricCommander database.
- The second phase is report generation: data collected previously is retrieved and organized into reports.

The report phases are separated so data is gathered once but then available to use in a variety of different reports either immediately or later. For example, one report might summarize errors within a particular job, and another report might display error trends from all jobs over the past month.

ElectricCommander implements data collection with a *postprocessor*.

- A postprocessor is a command associated with a particular procedure step. If a postprocessor is specified for a step, it executes concurrently with the main step command.
- The postprocessor runs on the same machine as the main command and in the same working directory, and it retrieves the log file from the step as its standard input.

ElectricCommander includes a standard postprocessor called *postp* that you can use and extend.

postp scans the step's log file looking for interesting output such as error messages and then sets properties on the job step to describe what it found. For example, *postp* might create a property named "errors" whose value is a count of the number of error messages in the log file, or a property named "tests" that counts the number of tests executed by the step. Also, *postp* can extract portions of the step log that contain useful diagnostic information and save this information for reporting.

Standard Commander reports, such as those on the Job Details page, display information collected by *postp* such as properties named "errors" and diagnostic log extracts. This information is available immediately, even before the step completes execution, so you can view it via ElectricCommander's web interface to monitor step execution. Also, you can create additional report generators of your own, which can use the same information displayed by ElectricCommander and/or any other additional information of your choice.

Postp

Commander's general-purpose postprocessor, *postp*, uses regular expression patterns to detect interesting lines in a step log.

- *postp* is already configured with patterns to handle many common cases such as error messages and warnings from *gcc*, *gmake*, *cl*, *junit*, and *cppunit*, or any error message containing the string "error." You can use the empty matcher group named "none" if just want to run a *postpEndHook*.
- *postp* is easy to use by simply setting the postprocessor for a step to "postp."
- *postp* also supports several useful command-line options.

To explore these options, invoke "postp --help" from your command-line.

Extending *postp*: matchers

If you find useful patterns in your log files undetected by *postp*, you can extend *postp* with additional patterns. This feature is easily implemented, but the extension interface currently involves writing simple Perl scripts and you need to look at the Perl source file for *postp* as you do this. The *postp* source code is installed during ElectricCommander installation and located in the `src/postp.pl` file in the distribution directory.

Postp is driven by a collection of *matchers*, which are regular expression patterns that select certain lines from step logs, and by a collection of Perl functions the matchers invoke to handle lines of interest. A matcher is a Perl hash with three values similar to the following example:

```
{
  id      => "error",
  pattern => q{ERROR:|[Ee]rror:},
  action  => q{
    incValue("errors"); diagnostic("", "error", -4)
  },
}
```

Explanation of the values in the Perl hash example above:

- **id** - A unique name for the matcher—used to identify the matcher in command-line arguments and a few other places.
- **pattern** - A regular expression tested against each line of the step log file. This particular pattern matches lines containing any of the strings "ERROR:", "Error:" or "error:"
- **action** - A Perl script that executes whenever a log line matches the pattern for this matcher. The script in this example increments a variable named "errors" that is copied automatically to a job step property with the same name. The script also saves a portion of the step log beginning 4 lines prior and extending through the current line and associates those lines with this error so it can be displayed in the web interface. See below for more information on the `incValue` and `diagnostic` functions.

To extend *postp* with patterns of your own, write a Perl script to add new patterns to the `@: :gMatchers` array. Here is a simple example:

```
my @newMatchers = (
  {
    id      => "coreDump",
    pattern => "core dumped",
    action  => q{
```

```
        incValue("coreDumps")
    }
},
{
    id      => "segFault",
    pattern => "segmentation fault",
    action  => q{
        incValue("segFaults")
    }
},
);
push @::gMatchers, @newMatchers;
```

These matchers detect lines containing the strings "core dumped" or "segmentation fault" and increment separate variables for each line type.

After writing extension code, you must ask `postp` to execute the code when it starts up. You can execute extension code in one of two ways:

- Place the code into a file and invoke `postp` with the `--load` option.
For example, `postp --load fileName`
- Copy the code into a property in `ElectricCommander` and use the `--loadProperty` option to `postp`.
For example, `postp --loadProperty /myProject/extraMatchers`

`Postp` extensions can contain arbitrary Perl code, which means you can use this mechanism to define additional functions to invoke in matcher actions if existing functions do not provide what you need.

Note: Additional `postp` matchers are available for your convenience. The matcher sample directory was installed during `ElectricCommander` installation. Go to `src/samples/postp`.

Postp functions

`Postp` contains several built-in functions to invoke in your matchers. The most useful functions are summarized below. Also, you can scan `postp` code for additional functions.

debugLog(format, arg, arg, ...)

Outputs information to the debug log, if the `--debugLog` command-line switch is set. Format provides a format string similar to `printf`, and each argument provides a value to substitute into the format string.

backTo(pattern, start)

This function searches backward to find the first line in the step log matching "pattern" (a regular expression) and returns the offset of that line relative to the current line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to -1).

backWhile(pattern, start)

This function searches backward to find the first line in the step log that does not match "pattern" (a regular expression) and returns the offset relative to the current line of the line just after the first non-matching line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to -1).

currentModule()

Returns the name of the current module (as determined by previous calls to `pushModule` and `popModule`), or an empty string if there is no current module.

diagnostic(name, type, first, last)

This function extracts a group of contiguous lines from the log file and saves them along with additional information for reports such as the log extracts, which appear at the bottom of the Job Details web page. The range of lines to extract is indicated by "first" and "last," each of which is an offset relative to the current line. For example, if "first" is -3 and "last" is 2, then 6 lines will be recorded: 3 lines before the current line, the current line, and 2 lines after the current line. In many cases, the values for "first" and "last" are computed by calling functions such as "forwardTo" or "backWhile." "Name" provides an identifier for this particular diagnostic, such as the name of a test that failed or a file that did not compile. "Type" specifies which kind of information this is, and must be "error," "warning," or "info." In addition to log lines, this function records "name," "type," the current module, if any, and the name of the current matcher.

forwardTo(pattern, start)

This function searches forward to find the first line in the step log matching "pattern" (a regular expression) and returns the offset of that line relative to the current line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to 1).

forwardWhile(pattern, start)

This function searches forward to find the first line in the step log that does not match "pattern" (a regular expression) and returns the offset relative to the current line of the line just before the first non-matching line. The result is normally used as an argument to the "diagnostic" function. "Start" is optional; it specifies the first line to check and is specified as an offset relative to the current line (it defaults to 1).

incValue(name, increment)

Adds "increment" to a value named "name" and arranges for that value to be written eventually to a property by the same name on the current job step. If this is the first call for "name," its value is initialized to 0. "Increment" is optional and defaults to 1.

Note: postp does not check the job step for a pre-existing property with the same name; it simply overwrites it.

logLine(lineNumber)

Returns the line from the step log given by lineNumber. 1 corresponds to the first line in the step log and the Perl variable `$_:gCurrentLine` holds the number of the current line. This function caches a sliding window of lines in the file, allowing you to go back to retrieve lines preceding the current line (as long as they do not precede it by too many lines). If the requested line is off the end of the file then `undef` is returned. If the requested line is before the beginning window of cached lines, an empty string is returned.

popModule()

Cancels the effect of the most recent call to `pushModule`, resetting the current module name to whatever it was before the corresponding call to `pushModule`.

postpEndHook()

If you define a function with this name, it invokes after `postp` has finished processing the log file, but before it makes its final properties update on the job step. Use this function to perform your own operations such as generating an error if the log file did not contain a particular line you were expecting.

pushModule(name)

In some situations it is possible to divide the log file into parts corresponding to different modules. For example, with recursive make invocations, there are typically notifications in the log output before and after each recursive make. This function is invoked to indicate a new module is being entered, where "name" is the name of the module. After this function is called, the "diagnostic" function will include "name" with error or warning messages to provide additional information in job reports. The previous module name, if any, is saved; you can return to it by calling `popModule`.

setProperty

This function sets a property in the ElectricCommander server. If the named parameter is a relative path, like `moduleCount`, the property is set or created on the current job step. You can use an absolute path, like `/myJob/fileLocation` also. Calling `setProperty` does not result in an immediate call to the Commander server. The property is added to an update list for updating at the next "update interval", typically every 30 seconds.

Integration with the Commander user interface

`postp` interacts with the ElectricCommander UI using two methods. The first method: Create custom properties with special names that are recognized by the UI itself. The second method: Create a file that contains "diagnostics", which are used to display errors or warnings, and link them to specific sections in the step's log file.

Custom property names and values

`postp` can be used to create properties in Commander, on the job step or anywhere else in Commander. However, the following properties are used by the standard ElectricCommander UI, so you should use these property names whenever possible, and avoid using these names in ways that conflict with the definitions below.

- **compiles** - the number of files compiled during the job step
- **diagFile** - the filename in the top-level directory of the job's workspace, containing diagnostics extracted from the step's log file
- **errors** - the number of errors (compilation failures, test failures, application crashes, and so on) that occurred during the job step. When property errors are set by `postp`, the step outcome is set to error also.
- **tests** - the number of tests executed by the job step, including successes and failures
- **testsSkipped** - the number of tests skipped during the job step
- **warnings** - the number of warnings that occurred during the job step. When property warnings is set by `postp`, the step outcome is set to warning also.

- **preSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *before* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **postSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *after* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **summary** - if this property exists, its value is displayed in the "Status" field (in the job reports) for this step, replacing whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

Diagnostic information

The second form of postprocessor generated output contains diagnostic extracts from the step's log file, typically providing additional information about problems. The postprocessor stores diagnostic information in an XML file in the top-level directory of the job workspace, then sets the step's **diagFile** property with the name of the file.

Diagnostic files must have a format like the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<diagnostics>
  <diagnostic>
    <matcher>compileError</matcher>
    <name>testa.c</name>
    <type>error</type>
    <module>util/timeLib</module>
    <firstLine>2</firstLine>
    <numLines>7</numLines>
    <message>testa.c: In function 'procl':
      testa.c:12: error: parse error before ';' token
      testa.c:13: error: 'for' loop initial declaration used outside C99 mode
      testa.c:14: error: parse error before ';' token
      testa.c:16: error: too few arguments to function 'exit'
      testa.c:18:2: warning: no newline at end of file
      testa.c:18: error: parse error at end of input
    </message>
  </diagnostic>
  <diagnostic>
    <name>testa.o</name>
    <type>error</type>
    <module>util/timeLib</module>
    <firstLine>9</firstLine>
    <numLines>1</numLines>
    <message>make: *** [testa.o] Error 1 </message>
  </diagnostic>
</diagnostics>
```

XML elements in the diagnostic file are as follows:

- **diagnostics** - Overall container; its children consist of all of the diagnostic elements.
- **diagnostic** - Describes one diagnostic extract; the elements described below are all children of this element.

- `matcher` - (optional) Identifier for the matcher that triggered this diagnostic; used primarily for debugging.
- `name` - (optional) Identifier that indicates the problem or situation that resulted in this diagnostic, such as the name of a failed test or the name of a file that did not compile.
- `type` - Type of message: must be "error," "warning," or "info."
- `module` - (optional) Name of the module in which the issue occurred, such as the name of a source code module being compiled at the time of a compile error.
- `firstLine` - Line number in the log file for the first line of the diagnostic extract (1 means the first line of the file). This element is used to provide a link from the diagnostic extract to the full log file.
- `numLines` - Total number of lines included in the diagnostic extract.
- `message` - The actual lines from the log file.

Postp integration with Java Tools

ElectricCommander integrates with three standard Java tools through postp matchers. These Java tools are:

- EMMA - an open source toolkit for measuring and reporting Java code coverage (Emma v2.0)
- JUnit - a framework for writing and running automated tests (tested with Ant v1.7)
- Clover - Atlassian's Java code coverage (generated by system, functional, or unit tests) analysis tool (Clover v2.4)

Note: If one of these Java tools is invoked in a Commander job step, Commander automatically detects the invocation [if you are using Postp] and adds any reports generated by these tools to the list of links at the top of the Job Details page.

The process...

Postp parses output from the invoked JAVA tool to match paths to reports it has already created. Then, postp copies all files that comprise the report to a unique location in the "Artifacts" Directory. Next, postp generates a link to the location in the Artifacts Directory to make the report available in the Links section on the Job Details page.

The following example of generated output from Emma illustrates this process:

```
emma:

init:
[mkdir] Created dir: C:\Documents and Settings\ptharani\Desktop\emma\emma-
2.0.5312\examples\out

compile:
[javac] Compiling 4 source files to C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\out

run:
[emmajava] EMMA: processing classpath ...
[emmajava] EMMA: [3 class(es) processed in 47 ms]
[emmajava] main(): running doSearch()...
[emmajava] main(): done
[emmajava] EMMA: writing [txt] report to [C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\coverage\coverage.txt] ...
```

```
[emmajava] EMMA: writing [html] report to [C:\Documents and
Settings\ptharani\Desktop\emma\emma-2.0.5312\examples\coverage\coverage.html] ...

all:

BUILD SUCCESSFUL
Total time: 1 second
```

From this output...

The actual link may reside here:

```
/home/commanderWorkspace/job_112_
200901081732/artifacts/javaTools/238/emmaCoverage/2/coverage.html
```

While the link name may be:

```
Step Id 238 ant-on-the-fly - emma report# 2
```

And the value of the link may be:

```
jobSteps/238//javaTools/238/emmaCoverage/2/coverage.html
```

Java Tool matcher examples

Two examples of postp Emma matchers:

```
{
  id      => "emmaReport1",
  pattern => q{EMMA: writing},
  action  => q{emmaExtractReport()},
},
{
  id      => "emmaReport2",
  pattern => q{\\[report\\] writing},
  action  => q{emmaValidateOutput()},
},
```

An example of postp JUnit matchers:

```
{
  id      => "junitReportCapture",
  pattern => q{\\[junitreport\\] Processing},
  action  => q{junitExtractReport ()},
},
```

These matchers correspond to the following output:

```
junit.report:
[junitreport] Processing
/net/WinStor2home/ptharani/junitDemo1/sample/testreport/TESTS-TestSuites.xml to
/tmp/null11214791178
[junitreport] Loading stylesheet jar:[file:/usr/local/tools/common/apache-ant-
1.7.0/lib/ant-junit.jar!/org/apache/tools/ant/taskdefs/optional/junit/xsl/junit-
frames.xsl
[junitreport] [file:/usr/local/tools/common/apache-ant-1.7.0/lib/ant-
junit.jar!/org/apache/tools/ant/taskdefs/optional/junit/xsl/junit-frames.xsl
[junitreport]] Transform time: 622ms
[junitreport] Deleting: /tmp/null11214791178
```

An example of postp Clover matchers:

```
{
  id      => "cloverHtmlReportAntTask",
  pattern => q{\\[clover-html-report\\] Writing HTML report to},
  action  => q{cloverExtractReport()},
},
```

These matchers correspond to the following output:

```
clover.report:
[clover-html-report] Clover Version 2.4.0, built on November 05 2008 (build-747)
[clover-html-report] Loaded from: /home/cloverDemo/clover-ant-2.4.0/lib/clover.jar
[clover-html-report] Clover: Evaluation License registered to electric cloud.
[clover-html-report] You have 27 day(s) before your license expires.
[clover-html-report] Loading coverage database from: '/home/cloverDemo/clover-ant-
2.4.0/tutorial/.clover/clover2_4_0.db'
[clover-html-report] Writing HTML report to '/home/cloverDemo/clover-ant-
2.4.0/tutorial/clover_html'
[clover-html-report] Done. Processed 1 packages in 2559ms (2559ms per package).
```

Artifacts directory

The value of the artifacts directory determines the scope of what is visible in the Commander UI from a job's workspace. By default, the Artifacts Directory is set to "artifacts", so the artifacts directory would have the form:

```
<path to job workspace>/artifacts
```

The value of the artifacts directory can be set in any of the following properties:

```
/myJob/artifactsDirectory
/myProject/artifactsDirectory
/server/settings/artifactsDirectory
```

Postp queries these properties [in the order listed] to determine the location of the Artifacts Directory. If no value is found (because the property was never set), the default "artifacts" is used.

Postp has a feature where it recognizes that data belongs to some standard tool (such as Junit), copies the logs produced by that tool to the artifacts directory, and then creates the report-url properties. These actions are done with the junitReportCapture matcher.

If you want to disable this matcher in your postp invocation in the step that runs junit tests, do the following:

```
postp --dontCheck junitReportCapture
```

It is possible that the artifacts directory will still be created even if nothing is put in it. If that occurs, set the artifactsDirectory property on your job (or the owning project) to empty-string.

Reports

ElectricCommander provides multiple reports and custom report capabilities to help you manage your build environment.

- Real-time reports - filtered view of your workload in real-time
- Build reports - summary reports produced at the end of a build and attached to the job
- Batch reports - summaries of your build environment with trends over time, two types:
 - Default Batch reports - automatically installed during ElectricCommander installation and scheduled to run daily
(Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)
 - Optional Batch reports - you can configure, rename, and schedule these reports to fit your requirements
(Category Report, Procedure Usage Report, Count Over Time Report, Multiple Series Reports)
- Custom reports - your choice to create and add at any time

Note: Batch and Custom reports must be run on the Commander server's agent (local agent) only. These reports need the BIRT report engine, which is installed on the Commander server.

Run reports on a non-local resource

If you want to configure reports to run on a non-local resource, do the following:

1. In the web interface, go to Projects > Electric Cloud > runReports > runEcrypteddata.
2. Change the Parameter Resource from the default value "local" to the name of a different resource. the resource must have access to the plugins directory.

Real-time reports

Home page

The Home page is a user-customizable dashboard that can be configured to show you just the Commander jobs you care about.

The Jobs Quick View section (on the right-side of the Home page) shows jobs you define using filters on job properties. You can create multiple categories filtered on specific project names, procedure names, users, or any other job property. Summary details for the job "pop-up" if you hover your mouse over one of the jobs.

Jobs

Unlike the Home page, which allows you to see specific subsets of work, the Jobs page shows all jobs that are running and all jobs that have completed. This is a good report to view when you need to see a list of all jobs run by any user or schedule. System administrators and managers can use this page to monitor overall system throughput.

Job details / Step details

After a job starts, you can get execution details from the Job Details page. To view job details, click on the job from the Home page or the Jobs tab. From within the Job Details page you can drill down to see details for an individual step. This is the finest grain of reporting, showing the detailed results of a specific job.

Resources

The Resources page reports the current state of your defined resources. This page also shows the current state of the agent (connected or not) and any steps currently running on that resource. Use this report to monitor your resources in near real time.

Build reports using *ecreport*

ElectricCommander includes a utility, *ecreport*, that summarizes build results and sends that summary through email. This customizable utility allows you to create a simple matrix based on results stored in properties and log files. Details such as SCM change logs and errors can be included.

In addition, you can attach your own reports to any job and have them show up as attachments to the Job Details Report. To attach custom reports, put them in the top-level workspace directory in a file that ends with ".html" and contains the string "report" or "Report." Names of all matching files are displayed in the Summary section in the upper right corner of the page. Click on a report name to view the report.

The following is an example of *ecreport* output:

thunder-main.4315-200706261344 is good!			View Online
	Windows	Linux	
Compile	1183 compiles	1121 compiles	
Build installer	ok	ok	
Agent unit tests	748 tests	642 tests	
Server unit tests	1875 tests	1860 tests	
Web unit tests	1556 tests	1557 tests	
Tool unit tests	410 tests	338 tests	
Install	ok	ok	
System tests	22 tests	22 tests	

General Information	
Start Time	6/26/2007 13:44:19
Elapsed Time	1 hour, 27 minutes, 6.0 seconds
Workspace	//f2/scratch/buildserver3build/thunder-main.4315-200706261344
Recent Updates	

----- chris -----

Change 24248 by chris@chris-thunder on 2007/06/26 13:24:31

Adding "index-redirect.php", which will be used to redirect web browsers to the "thunder" sub-directory when users only type in the host name for the web server. At install time, this file should be moved to the apache/htdocs directory, and renamed "index.php".

This will resolve the issue of https redirection not working when using the httpd.conf "Redirect" directive.

Jobs fixed ...

NMB-2761 on 2007/06/26 by tom *closed*

<https://buildserver2:443> redirects to <http://www.thunder.com/>

Affected files ...

.. //depot/thunder/main/web/docroot/index-redirect.php#1 add

ecreport extracts information about a particular job from the Commander server and writes an HTML report to standard output. The report is organized as a 2-dimensional table summarizing the results of steps in the job. The table layout can be customized by defining its format using the `ectool` (or `ec-perl`) `--load` option. Invoke the program in the top-level directory of the job's workspace so it can access log file extracts in that workspace (or, use the `--dir` option).

The following options are available for use with the *ecreport* utility:

Option	Description
<code>--culprit</code>	If specified and the job failed, the report will be emailed to each user mentioned in the updates file. The value of this option provides the subject line for the message (%s will be replaced with the job name).
<code>--data</code>	Name of file containing job data for the report. If specified, this data is used in place of querying the Commander server. Used primarily for testing.
<code>--dir</code>	Change to this working directory before doing anything else.
<code>--help</code>	Print this message and exit without doing anything.
<code>--jobId</code>	Identifier for the Commander job to report on (required unless <code>--data</code> is supplied). This is a UUID.
<code>--load</code>	Name of a file containing Perl code to evaluate after option processing. Used primarily for debugging. There can be multiple <code>--load</code> options.
<code>--emailConfig</code>	The name of the Email Configuration to use when emailing the report. Defaults to "default".
<code>--mailto</code>	Email the report to this address(s).
<code>--replyto</code>	Value for the "Reply-To" field in emailed reports.
<code>--server</code>	Location of the Commander server (hostname or hostname:port), to retrieve job data. Defaults to the <code>COMMANDER_SERVER</code> environment variable.
<code>--subject</code>	Value for the "Subject" field in emailed reports. If the value contains a %s, a string summarizing the job replaces the %s.

Option	Description
<code>--updates</code>	Name of a file containing information about recent updates reflected in the job. If specified, the contents of this file are included in the report.
<code>--version</code>	Print ecreport version number.
<code>--webRoot</code>	Base URL for the Commander Web server where the job was run, such as <code>http://myServer</code> . URLs in the report will refer to ElectricCommander Web pages relative to this URL.

Example 1

```
ecreport --jobId $[/myjob/jobId] --load ./rptformat.pl --updates update.log --webRoot
```

```
http://myServer/commander > ecreport.html
```

This example writes a report to the file `ecreport.html`. Because this file has an `.html` extension, it will be displayed in the Reports section at the top of the Job Details page. The format of the table for this report is specified in `rptformat.pl`. An example of the contents from this file would be:

```
@:gTableRows = (  
  ["",  
    "Windows",  
      "Linux",  
      "Solaris"],  
  ["Extract",  
    "span",  
    "span",  
    "update.log"],  
  ["Compile",  
    "windows-compile.log",  
    "linux-compile.log",  
    "solaris-compile.log"],  
  ["Unit test",  
    "windows-unittest.log",  
    "linux-unittest.log",  
    "solaris-unittest.log"],  
  ["System test",  
    "windows-systemtest.log",  
    "linux-systemtest.log",  
    "solaris-systemtest.log"]  
);
```

The “Recent Updates” report section is specified in `update.log`. The `--webRoot` argument defines the root for any URL that appears in the report.

Example 2

```
ecreport --jobId $[/myJob/jobName] --load ./ecreportConfig.pl --mailto  
user1@company.com --replyto user2@company.com  
--subject 'Build Report' --updates updates.log  
--webRoot http://myServer/commander
```

This report will be sent as the body of an email message to `user1@company.com` with subject “Build Report”.

Default Batch reports (Run Reports)

These reports are installed during the ElectricCommander installation and automatically scheduled to run daily.

To access these reports: Select the Projects tab, then select the Electric Cloud project, then click the Reports subtab. This report collection graphs trends over time, including success/failure, build times, and resource consumption. These reports allow you to customize two levels of grouping into "variants." These variants may represent products, branches, locations, architectures, or any other descriptors you would like to use to group your workload.

Note: After a Default Batch Report runs, a link for each report appears on the Job Details page in the Links section. When you select a report from the Links section, it will be displayed in full-screen view.

Report types

Cross Project Summary - This report shows the status of multiple variants over the past 30 days. This high-level summary of each of the 30 days shows green, yellow, or red depending on the result of the best build of the day. The summary also shows average times, total invocations, and other data.

Variant Trend - This report shows more details about a particular variant such as build outcome over time, build quantity over time, elapsed build times over time, and a list of actual jobs that ran during the period.

Daily Summary - This report shows the most recent results of each variant, including the last successful build.

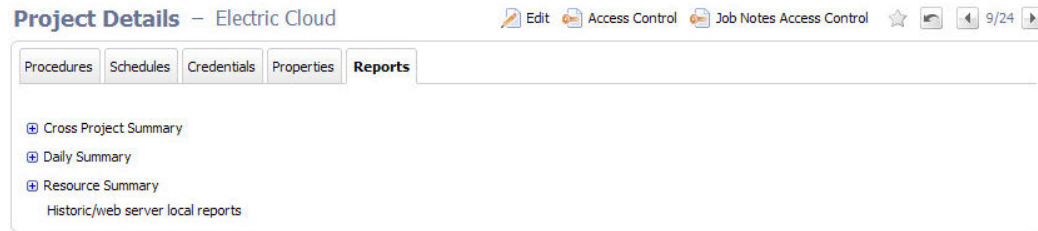
Resource Summary - This report shows resource utilization over the last 24 hours. You can see which resources are idle and which resources are busy. Commander has advanced features that allow you to select resources for your job at run time, using late binding of property names. This feature provides unparalleled flexibility and control over your environment, but could make it difficult to find out which resources were used. This report provides the visibility you need into resource loading.

Resource Detail - This report shows the utilization of a particular resource over the course of one day.

Wait Time - This report shows cumulative wait time due to license, resource, and workspace waits. It also shows the number of steps that were delayed for any of those reasons.

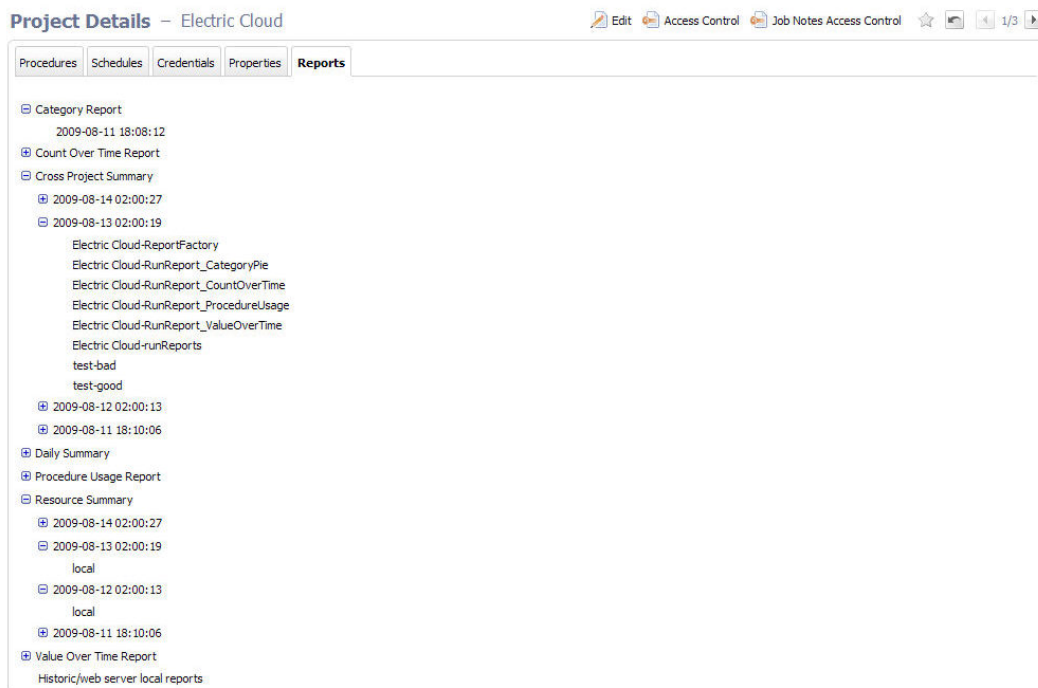
Default Batch Reports

- Cross Project Summary
 - Variant Trend
- Daily Summary
- Resource Summary
 - Resource Detail



The navigation tree on the left-side of the screen allows you choose the report and date of interest. These reports, which show long term trends over all projects and resources, are run from the `runReports` procedure in the Electric Cloud project.

The following screen example displays the Default Batch Reports, the historical reports link, and Optional Batch Reports (described below). Notice that you can expand a selected report to three levels. The third level represents drill-down links available within that report. In addition, if you click on a report name, the most current report is displayed.



Tip...

You may change the number of reports displayed under each report type. Locate the `propertySheet` named `ec_reportConfiguration` on the project or server (project overrides server) and find the "groupings" section. Next, you can create nested property sheets under `ec_reportConfiguration` with the properties `groupName`, `sortDirection`, and `limit`. If you set this information using `ectool`, you will see it on Commander's web interface also.

If the `ec_reportConfiguration` propertySheet does not exist, you will need to create it, then copy the example below into the propertySheet.

Example:

```
ectool setProperty ec_reportConfiguration/reportDaysLimit --value 90 --projectName foo

ectool setProperty ec_reportConfiguration/groupings/1/groupingName --value reportType
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/1/sortDirection --value ascending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/1/limit --value 100 --projectName
foo

ectool setProperty ec_reportConfiguration/groupings/2/groupingName --value reportDate
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/2/sortDirection --value descending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/2/limit --value 5 --projectName
foo

ectool setProperty ec_reportConfiguration/groupings/3/groupingName --value
reportResource --projectName foo
ectool setProperty ec_reportConfiguration/groupings/3/sortDirection --value ascending
--projectName foo
ectool setProperty ec_reportConfiguration/groupings/3/limit --value 100 --projectName
foo
```

This example explicitly defines the default behavior for project foo's reports.

The 1, 2 and 3 propertySheets define the order of the groupings. The actual names 1, 2, 3, could be anything as long as they are listed alphanumerically in the order you need.

Additionally, any groupings that do not exist will be skipped during output. For example, if you added a `reportRunByUser` grouping #4 and added it to some reports that define `reportType="User Builds"`, `reportDate`, and `reportRunByUser` properties (but not `reportResource`), you would get something similar to a resource usage report structure with users rather than resources at the deepest level.

Note: If you choose to display 500 or more reports at one time, you may experience some performance degradation.

Advanced reporting information - ecrptdata

How are default batch reports generated?

Reports are scheduled as part of a special procedure, `runReports`, in the Electric Cloud project. This procedure has one step, `runEcrptdata`, that runs the `ecrptdata` program. This program reads data from Commander, summarizes it for reports, and generates HTML reports using the BIRT reporting engine. The program takes input parameters. During installation, reporting is configured as follows:

```
ecrptdata --end yesterday --grain 300 --debug 2 --reports all
```

By default, all reports are run every day at 2am. The Cross Project Summary and Variant Trend shows data from the previous 30 days. The Daily Summary, Resource Summary, and Resource Detail reports show data from

the last full day (yesterday). You can adjust the schedule to run more often or at different times by altering the "Report Schedule" schedule in the Electric Cloud project.

Changing the report time period

If you would like to alter the report time period, you must change the input parameters to the `ecrptdata` program within the procedure step "Electric Cloud:runReports:runEcrptdata". Select the `Ecrptdata` step name to go to the Edit Step page. Make the changes you need in the Parameters section. If the `--end` option is omitted, the end of the reporting period is the time when the report runs (now). If the keyword "yesterday" is provided, reports will include full data for yesterday. Otherwise, a specific date and time can be given in the format "YYYY-MM-DDTHH:MM:SS".

Changing the report grouping

You can tell ElectricCommander how to group the Cross Project Summary and Variant Trend reports. By default, the outer grouping is "Project Name" and the inner grouping is "Procedure Name." To change these groupings, use the following `ecrptdata` options:

<code>--group1Name xxxx</code>	Set the column header for the first column of the Cross Project Summary report to "xxxx".
<code>--group1Property yyyy</code>	Use property "yyyy" as the name to group on in the first column of the Cross Project Summary report.

All lookups are done with respect to an individual job and will be converted to absolute property paths. The values `projectName`, `procedureName`, and `jobId` are set in context for the job being examined.

Examples:

`/myProject/foo` will be converted into the API call `"ectool getProperty /projects/projectName/foo"`

`/myProcedure/foo` will be converted into the API call `"ectool getProperty /projects/projectName/procedures/procedureName/foo"`

`/myJob/foo` will be converted into the API call `>"ectool getProperty /jobs/jobId/foo"`

`/myParameter/foo` will be converted into the API call `"ectool getActualParameter foo -jobId jobId"`

To show the value of the procedure name as the outer grouping and the value of the parameter "branch" as the inner grouping,

use the following `ecrptdata` options:

```
--group1Name Procedure --group1Property "/myProcedure/procedureName"
--group2Name Branch --group2Property /myParameter/branch
```

Note: The `/myParameter` "shortcut" to the property path was created specifically for `ecrptdata` only—it cannot be used as other `/my` property shortcuts are used. For more information on property shortcuts, see the [Properties](#) help topic. In addition, when working with `ecrptdata`, any `/my` value is case-insensitive. For example, you can use `/myParameter` or `/myparameter`.

Filtering

ecrptdata supports "findObjects" type filters when searching for jobs and job steps.

This feature is implemented with two new command line switches: "--jobFilter <property, operator, operand>" and "--stepFilter <property, operator, operand>". The jobFilter option filters the CrossProjectSummary, DailySummary, and VariantTrend reports. The stepFilter option filters the ResourceSummary and ResourceDetail reports.

Examples:

```
--jobFilter "branch,isNotNull,1"
--jobFilter "jobName,like,my-procedure%"
--stepFilter "outcome,notEquals,aborted"
```

Adding additional data columns to the report

You can add custom data columns to each report. These columns are specified by providing a comma separated list of property references in the following ecrptdata options:

--crossProjectProperties	sets extra columns for the Cross Project Summary Report
--variantTrendProperties	sets extra columns for the Variant Trend Report
--dailySummaryProperties	sets extra columns for the Daily Summary Report
--resourceSummaryProperties	sets extra columns for the Resource Summary Report
--resourceDetailProperties	sets extra columns for the Resource Detail Report

You must choose properties appropriate for the type of report and grouping you have chosen. For example, if you choose Project Name and Procedure Name for your groupings in the Cross Project Summary report, each line of the report will be in the context of a project or procedure. You can add extra columns to show properties on the project or procedure, but it would not make sense to show properties on a resource, job, or job step. For resource reports, the properties must be in the context of a resource.

The column header is the name of the property.

Examples:

```
--crossProjectProperties "/myproject/QA_State,/myprocedure/PartNumber"
--resourceSummaryProperties "/myresource/OS,/myresource/Architecture"
```


Secure login

`ecrptdata` provides options for secure login to the Commander server.

Currently, there are three ways to login:

1. Use both the "`--user <user>`" and "`--pass <password>`" options.
This method is not secure... the password will be in plain text and visible to others.
2. If you omit the "`--pass <password>`" option, the password will be read from `stdin`. You can then read the password from a secure file or from a stored credential.
For example:

```
cat mySafePasswordFile.txt | ecrptdata --user <user> .....  
ectool getFullCredential myCred --value password | ecrptdata --user <user> .....
```
3. Another option, "`--credential <credentialName>`", allows you to specify a credential name. If you use this option, "`--user <user>`" and "`--pass <password>`" are ignored. Note that the credential must be attached to the `runReports` job step. Both the user and password are retrieved from the credential.

Logic:

1. If `--credential` is used, attempt to lookup credential and extract user and password
2. If user is not set, return error
3. If password is not set, read from `stdin`
4. Attempt to login to server specified in `--server <server>` option using user and password

Optional Batch reports

Additional reports are available, but these reports are not scheduled to run by default. You can schedule these reports to suit your purpose. **Note:** When you run an Optional Batch Report, a link for each report appears on the Job Details page in the Links section. When you select a report from the Links section, it will be displayed in full-screen view. The following optional reports are available:

- Category Report
- Procedure Usage Report
- Count Over Time Report
- Multiple Series Report
(formerly known as the "Value Over Time" report)

These reports are available from the Reports tab on the Project Details page for your Project.

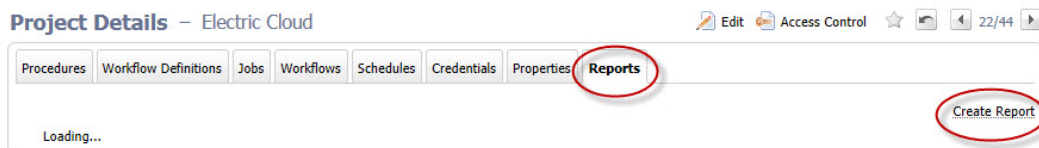
Note: For the Category, Count Over Time, and Multiple Series reports, a **View Search Results** link is available at the bottom of the report if a Saved Filter was used to generate the report.

Creating a report job

ElectricCommander runs reports as a job, so reports can be managed like any other job.

To create a report for your project:

1. Go to the Project Details page for your selected project.
2. Select the Reports subtab.
3. Click the **Create Report** link.



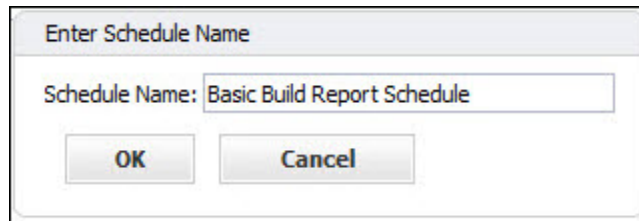
After clicking **Create Report**, the next screen is displayed.

Fill-in the fields or select the appropriate values as follows:

- Report Title - Your report title. Type over the default report name, choosing any unique name for your report. We supplied Basic Build Report for our report name.
- Saved Filter:
 - Project - Click your mouse inside this field to see a list of projects from which to make a selection. We select our FirstNewProject project name.
 - Filter - Use the BasicBuildFilter we created earlier.
- Time Period - Use the drop-down menu to select the time period.
- Create thumbnail? - Check this box so you can view this report on your Home page.
(We will create a "thumbnail" report view on your Home page at the end of this scenario.)
- Object Type - Use the drop-down menu to select Job for this example.
- Table column choices:
 - Chart Type - Use the drop-down menu to choose the chart type.
 - Function - Use the drop-down menu to choose the function you need.
 - Property Name - Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. We chose "outcome" for this example.
 - Display Name - You can choose a different unique Display Name if you prefer to do so.
 - Stacked - Select this checkbox to see your report results "stacked" versus overlaid.
 - The "X" icon - Click this icon to delete any row you no longer need.
- Select the **Add Series** button if you would like to create additional table entries for additional report information.
- Chart Options - Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.

Buttons at the bottom of the page:

- Run Report button - this button takes you to the Job Details page to run the report immediately—one time only.
- Click the **Create Schedule** button - this button displays a box with a default schedule name you can change.

A screenshot of a dialog box titled "Enter Schedule Name". It contains a text input field with the label "Schedule Name:" and the text "Basic Build Report Schedule". Below the input field are two buttons: "OK" and "Cancel".

Enter Schedule Name

Schedule Name: Basic Build Report Schedule

OK Cancel

- Supply a name for the schedule—again, we tied the schedule name to the procedure name for which we want the report.

Viewing the report

After the report job runs, view the report by selecting the Reports subtab within the project. **Note:** All reports are viewed from within a project context. If you have reports that span multiple projects, run them from the Electric Cloud project or create a project to store summary reports.

After selecting the Reports subtab, see the navigation tree on the left-side of the page. This tree allows you to select the report you want to view. The report tree contains three levels.

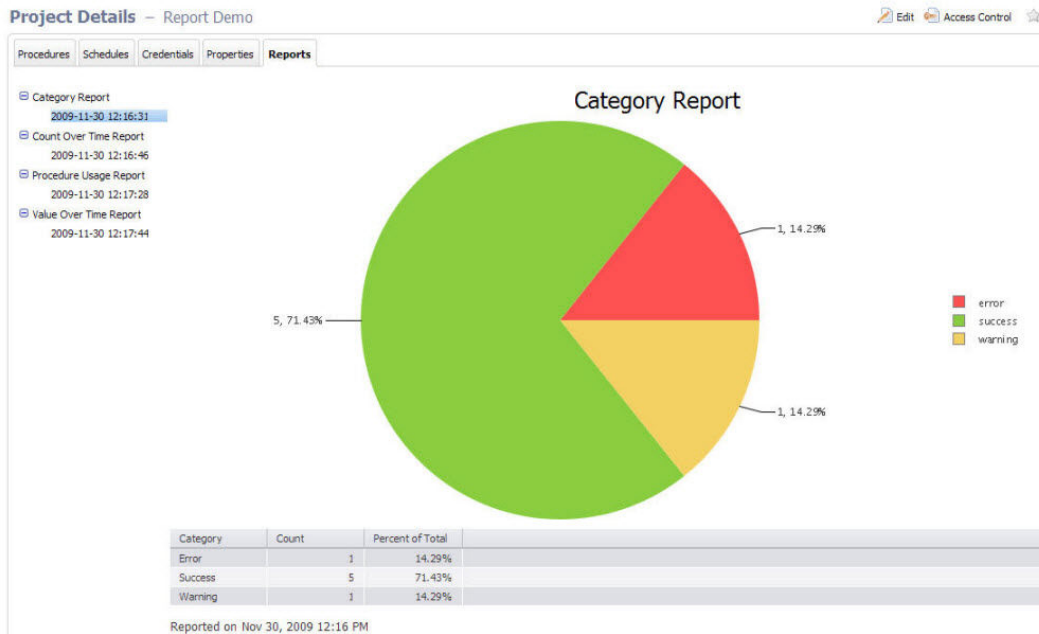
- Report Title - Click the "+" icon next to the report type to see available report versions.
- Report Date - Click this link to see report details.

Note: Click the report type to display the most recent report version.

Optional Batch report examples

Category sample report

This report shows the distribution of property values in a pie chart.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_CategoryPie

Parameters

Credential: Username: Password:

Locales:

Object Type:

Property: Default: "outcome"

Report Title: Default: "Category Report"

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Time Period:

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Run Cancel

Field descriptions

Credential - these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales -a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en, zh , ja, or ko, which translate respectively to English, Chinese,

Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. Note: Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Property - the property of interest (default outcome) and must be a valid property for the object type selected

Report Title - your report title (default Category Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select a time period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Procedure Usage sample report

For a specific procedure, this report shows procedures called by the procedure and which procedures it called.

Project Details – Report Demo

Edit Access Control Job Notes Access Control

Procedures Schedules Credentials Properties **Reports**

Category Report
2009-01-12 08:04:57.6
Count Over Time Report
2009-01-12 08:05:17.7
Procedure Usage Report
2009-01-12 08:17:05.0
Value Over Time Report
2009-01-12 08:07:14.9

Procedure Usage Report

Project	Procedure	Calls/Called By	Project	Procedure
My Library	LibProc-Bar	called by	My Library	LibProc-Foo
My Library	LibProc-Foo	calls	My Library	LibProc-Bar
My Library	LibProc-Foo	called by	Sample Project	Do Work

Reported on Jan 12, 2009 8:16 AM

Creating this report on the Run Procedure page...

Run Procedure – RunReport_ProcedureUsage

Parameters

Credential: Username: Password:

Locales:

Project: Browse

Project Procedure: Required; Default: "%"

Report Title: Default: "Procedure Usage Report"

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Run Cancel

Field descriptions

Credential - these are the credentials to use for collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en , zh , ja , or ko, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Project - this is the name of the project where you want to install the report.

Project Procedure - enter a project procedure. "called" and "called by" data is shown for that project/procedure combination. You can use "wildcards." For example, to see data for ALL projects and procedures, enter the % character in each field.

Report Title - this is the title of the report

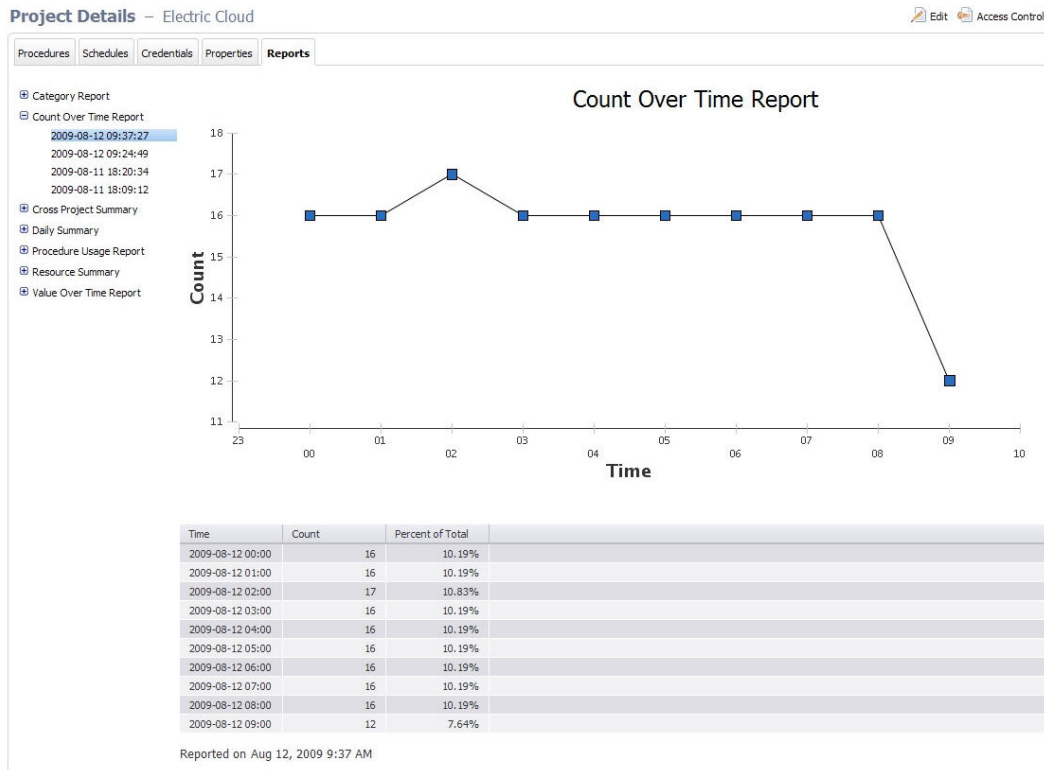
Count Over Time sample report

This report shows search results from counting multiple job, job step, or resource objects within the selected time range.

For example, this report counts the number of occurrences for the specified object-type over the selected time range--if a filter exists, it will be applied. This report does not aggregate the value (use ValueOverTime if you want aggregation), each occurrence counts as one.

Use this report when the property contains string values. The ValueOverTime report can be used only for numeric properties.

Note: Generating large numbers of datapoints can exceed BIRT's charting capability.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_CountOverTime ☆

Parameters

Chart Time Grouping: Default: "Auto"

Chart Type: Default: "Line"

Chart X-Axis Label: Default: "Time"

Chart Y-Axis Label: Default: "Count"

Credential: Username: Password:

Locales:

Object Type: Default: "Job"

Report Title: Default: "Count Over Time Report"

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Table Column Heading Property: Default: "Count"

Table Column Heading Time: Default: "Time"

Table Time Grouping:

Time Period:

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Field descriptions

Chart Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, or Year

Chart Type - these are: Line (default), Bar, Cone, Tube, Triangle, or Area

Chart X-Axis Label - the text to display on chart x-axis (default Date)

Chart Y-Axis Label - the text to display on chart y-axis (default Count)

Credential - credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en, zh, ja, or ko, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Report Title - your report title (default Count Over Time Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Table Column Heading Property - the table heading for the right column (default Count)

Table Column Heading Time - the table heading for the left column (default Date)

Table Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, Year

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

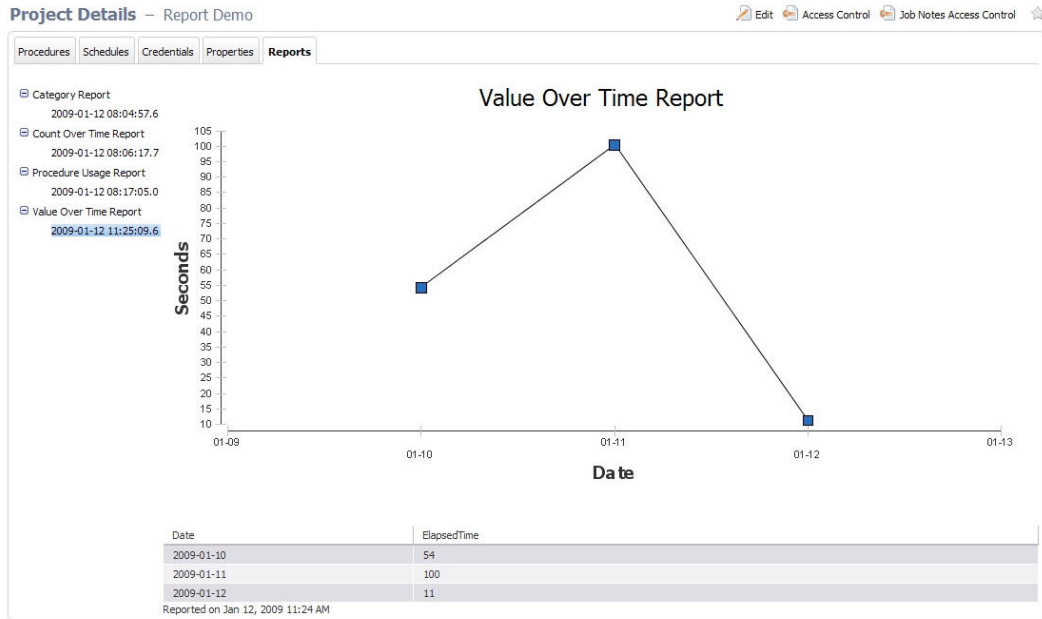
This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Multiple Series report

This reports shows the cumulative value of a property over time. This report should be run against properties that contain numbers. (Use the Count Over Time report if your property contains string values.) You can specify the function to use for aggregating values (sum, count, average). For example: Assume a property that contains the number of failed unit tests is stored on each job. Using the Multiple Series report, you can see the failure trend over time.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_ValueOverTime

Parameters

Chart Time Grouping: Default: "Date"

Chart Type: Default: "Seconds"

Chart X-Axis Label: Default: "Date"

Chart Y-Axis Label: Default: "Seconds"

Credential: Username: Password:

Locales:

Object Type: Default: "elapsedTime"

Property: Default: "elapsedTime"

Property Expression:

Property Function: Default: "Value Over Time Report"

Report Title:

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Table Column Heading Property:

Table Column Heading Time: Default: "Date"

Table Time Grouping:

Time Period:

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Field descriptions

Chart Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, or Year

Chart Type - these are: Line (default), Bar, Cone, Tube, Triangle, or Area

Chart X-Axis Label - the text to display on chart x-axis (default Date)

Chart Y-Axis Label - the text to display on chart y-axis (default Count)

Credential - these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: `en`, `zh`, `ja`, or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Property - the property of interest (default outcome) and must be a valid property for the object type selected

Property Expression - an optional mathematical function to apply to your data. For example, elapsed time is recorded in milliseconds. To make a useful report with elapsed times, you may want to show seconds, so this field would be entered as `"/1000"`.

Property Function - the following list contains the aggregation functions to use for data in the chart and table.

Count	Median
Count Distinct	Mode
First	Std Deviation
Last	Sum
Min	

Report Title - your report title (default Value Over Time Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Table Column Heading Property - the table heading for the right column (default Count)

Table Column Heading Time - the table heading for the left column (default Date)

Table Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, Year

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Optional Batch reports

Additional reports are available, but these reports are not scheduled to run by default. You can schedule these reports to suit your purpose. **Note:** When you run an Optional Batch Report, a link for each report appears on the Job Details page in the Links section. When you select a report from the Links section, it will be displayed in full-screen view. The following optional reports are available:

- Category Report
- Procedure Usage Report
- Count Over Time Report
- Multiple Series Report
(formerly known as the "Value Over Time" report)

These reports are available from the Reports tab on the Project Details page for your Project.

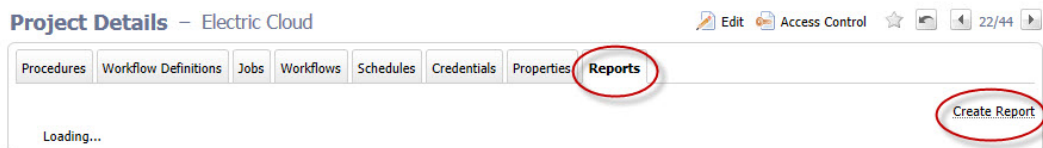
Note: For the Category, Count Over Time, and Multiple Series reports, a **View Search Results** link is available at the bottom of the report if a Saved Filter was used to generate the report.

Creating a report job

ElectricCommander runs reports as a job, so reports can be managed like any other job.

To create a report for your project:

1. Go to the Project Details page for your selected project.
2. Select the Reports subtab.
3. Click the **Create Report** link.



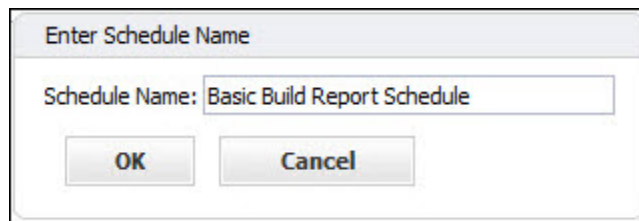
After clicking **Create Report**, the next screen is displayed.

Fill-in the fields or select the appropriate values as follows:

- Report Title - Your report title. Type over the default report name, choosing any unique name for your report. We supplied Basic Build Report for our report name.
- Saved Filter:
 - Project - Click your mouse inside this field to see a list of projects from which to make a selection. We select our FirstNewProject project name.
 - Filter - Use the BasicBuildFilter we created earlier.
- Time Period - Use the drop-down menu to select the time period.
- Create thumbnail? - Check this box so you can view this report on your Home page.
(We will create a "thumbnail" report view on your Home page at the end of this scenario.)
- Object Type - Use the drop-down menu to select Job for this example.
- Table column choices:
 - Chart Type - Use the drop-down menu to choose the chart type.
 - Function - Use the drop-down menu to choose the function you need.
 - Property Name - Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties. We chose "outcome" for this example.
 - Display Name - You can choose a different unique Display Name if you prefer to do so.
 - Stacked - Select this checkbox to see your report results "stacked" versus overlaid.
 - The "X" icon - Click this icon to delete any row you no longer need.
- Select the **Add Series** button if you would like to create additional table entries for additional report information.
- Chart Options - Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.

Buttons at the bottom of the page:

- Run Report button - this button takes you to the Job Details page to run the report immediately—one time only.
- Click the **Create Schedule** button - this button displays a box with a default schedule name you can change.

A screenshot of a dialog box titled "Enter Schedule Name". It contains a text input field with the label "Schedule Name:" and the text "Basic Build Report Schedule". Below the input field are two buttons: "OK" and "Cancel".

Enter Schedule Name

Schedule Name: Basic Build Report Schedule

OK Cancel

- Supply a name for the schedule—again, we tied the schedule name to the procedure name for which we want the report.

Viewing the report

After the report job runs, view the report by selecting the Reports subtab within the project. **Note:** All reports are viewed from within a project context. If you have reports that span multiple projects, run them from the Electric Cloud project or create a project to store summary reports.

After selecting the Reports subtab, see the navigation tree on the left-side of the page. This tree allows you to select the report you want to view. The report tree contains three levels.

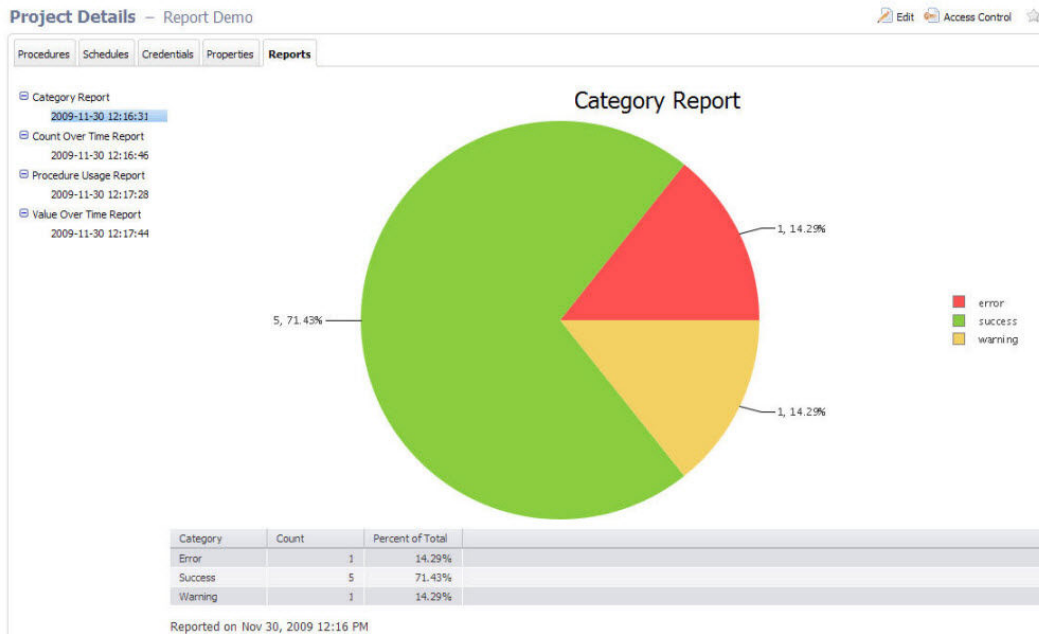
- Report Title - Click the "+" icon next to the report type to see available report versions.
- Report Date - Click this link to see report details.

Note: Click the report type to display the most recent report version.

Optional Batch report examples

Category sample report

This report shows the distribution of property values in a pie chart.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_CategoryPie

Parameters

Credential: Username: Password:

Locales:

Object Type:

Property: Default: "outcome"

Report Title: Default: "Category Report"

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Time Period:

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Run Cancel

Field descriptions

Credential - these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales -a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en, zh , ja, or ko, which translate respectively to English, Chinese,

Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. Note: Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Property - the property of interest (default outcome) and must be a valid property for the object type selected

Report Title - your report title (default Category Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select a time period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Procedure Usage sample report

For a specific procedure, this report shows procedures called by the procedure and which procedures it called.

Project Details – Report Demo

Procedures Schedules Credentials Properties **Reports**

Category Report
2009-01-12 08:04:57.6
Count Over Time Report
2009-01-12 08:05:17.7
Procedure Usage Report
2009-01-12 08:17:05.0
Value Over Time Report
2009-01-12 08:07:14.9

Procedure Usage Report

Project	Procedure	Calls/Called By	Project	Procedure
My Library	LibProc-Bar	called by	My Library	LibProc-Foo
My Library	LibProc-Foo	calls	My Library	LibProc-Bar
My Library	LibProc-Foo	called by	Sample Project	Do Work

Reported on Jan 12, 2009 8:16 AM

Creating this report on the Run Procedure page...

Run Procedure – RunReport_ProcedureUsage

Parameters

Credential: Username: Password:

Locales:

Project: Browse

Project Procedure: Required; Default: "%"

Report Title: Default: "Procedure Usage Report"

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Field descriptions

Credential - these are the credentials to use for collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en , zh , ja , or ko, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Project - this is the name of the project where you want to install the report.

Project Procedure - enter a project procedure. "called" and "called by" data is shown for that project/procedure combination. You can use "wildcards." For example, to see data for ALL projects and procedures, enter the % character in each field.

Report Title - this is the title of the report

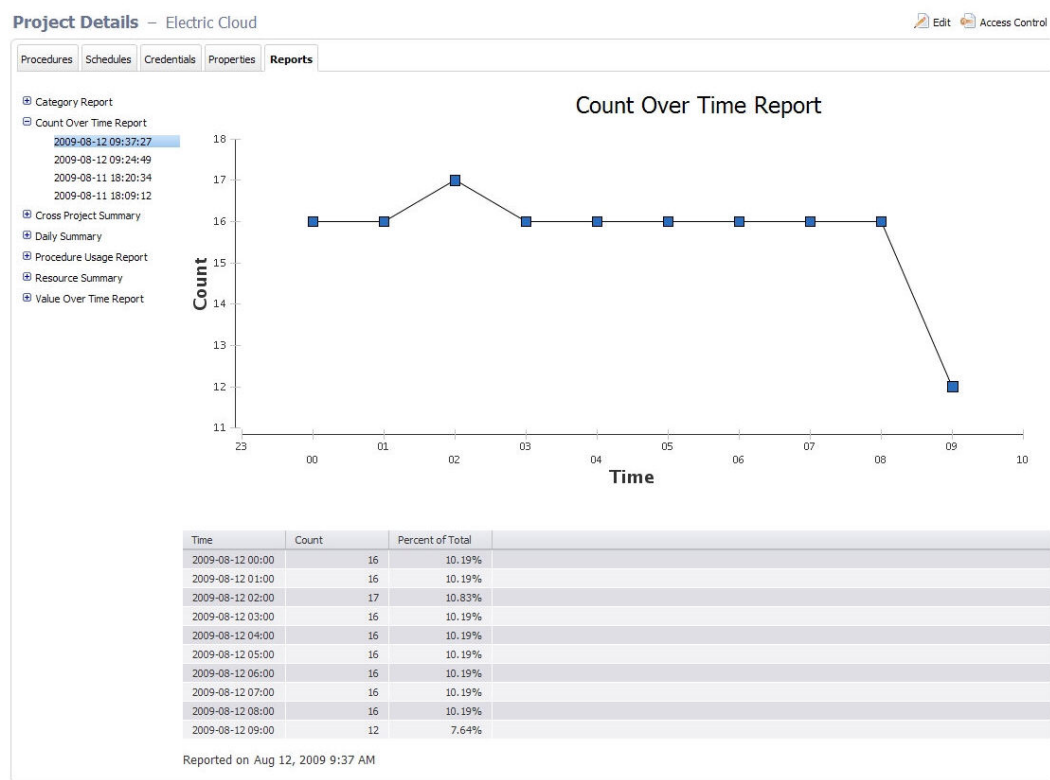
Count Over Time sample report

This report shows search results from counting multiple job, job step, or resource objects within the selected time range.

For example, this report counts the number of occurrences for the specified object-type over the selected time range--if a filter exists, it will be applied. This report does not aggregate the value (use ValueOverTime if you want aggregation), each occurrence counts as one.

Use this report when the property contains string values. The ValueOverTime report can be used only for numeric properties.

Note: Generating large numbers of datapoints can exceed BIRT's charting capability.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_CountOverTime ☆

Parameters

Chart Time Grouping: Default: "Time"

Chart Type: Default: "Count"

Chart X-Axis Label: Default: "Time"

Chart Y-Axis Label: Default: "Count"

Credential: Username: Password:

Locales:

Object Type: Default: "Count Over Time Report"

Report Title: Default: "Count Over Time Report"

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Table Column Heading Property: Default: "Count"

Table Column Heading Time: Default: "Time"

Table Time Grouping:

Time Period:

Advanced

Priority:

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Field descriptions

Chart Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, or Year

Chart Type - these are: Line (default), Bar, Cone, Tube, Triangle, or Area

Chart X-Axis Label - the text to display on chart x-axis (default Date)

Chart Y-Axis Label - the text to display on chart y-axis (default Count)

Credential - credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: en, zh, ja, or ko, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Report Title - your report title (default Count Over Time Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name blank, no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Table Column Heading Property - the table heading for the right column (default Count)

Table Column Heading Time - the table heading for the left column (default Date)

Table Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, Year

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

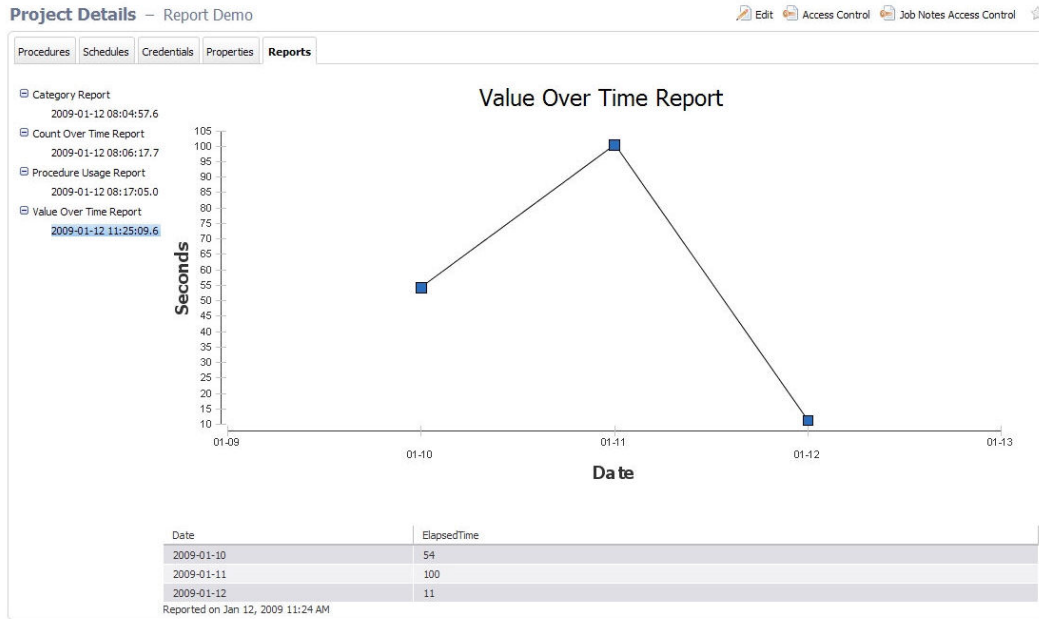
This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Multiple Series report

This reports shows the cumulative value of a property over time. This report should be run against properties that contain numbers. (Use the Count Over Time report if your property contains string values.) You can specify the function to use for aggregating values (sum, count, average). For example: Assume a property that contains the number of failed unit tests is stored on each job. Using the Multiple Series report, you can see the failure trend over time.



Creating this report on the Run Procedure page...

Run Procedure – RunReport_ValueOverTime

Parameters

Chart Time Grouping: Auto

Chart Type: Line

Chart X-Axis Label: Date Default: "Date"

Chart Y-Axis Label: Seconds Default: "Seconds"

Credential: Username: Password:

Locales:

Object Type: Job

Property: elapsedTime Default: "elapsedTime"

Property Expression:

Property Function: Average

Report Title: Value Over Time Report Default: "Value Over Time Report"

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Table Column Heading Property:

Table Column Heading Time: Date Default: "Date"

Table Time Grouping: Auto

Time Period: Yesterday

Advanced

Priority: normal

Impersonation: ☒ Use pre-defined credential ☐ Use specific credential ☐ Use a specific user

Run Cancel

Field descriptions

Chart Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, or Year

Chart Type - these are: Line (default), Bar, Cone, Tube, Triangle, or Area

Chart X-Axis Label - the text to display on chart x-axis (default Date)

Chart Y-Axis Label - the text to display on chart y-axis (default Count)

Credential - these are credentials to be used when collecting data. If no credentials are specified (Username/Password remain blank), the report runs with the privileges of the project from where it is run. If credentials are supplied, the data is collected using the Commander user security access in the credential. Use credentials when you want to access data outside of your project.

Locales - a comma separated list of locale(s) where you would like to render the report. Locale is specified as one or more of the following: `en`, `zh`, `ja`, or `ko`, which translate respectively to English, Chinese, Japanese, or Korean. If no locale is specified, the default is to use the locale of the Commander server. For example, a server already set for Chinese will see reports in Chinese with no other changes required. **Note:** Currently, Commander supports four locales only.

Object Type - the type of object to search (default job)

Property - the property of interest (default outcome) and must be a valid property for the object type selected

Property Expression - an optional mathematical function to apply to your data. For example, elapsed time is recorded in milliseconds. To make a useful report with elapsed times, you may want to show seconds, so this field would be entered as `"/1000"`.

Property Function - the following list contains the aggregation functions to use for data in the chart and table.

Count	Median
Count Distinct	Mode
First	Std Deviation
Last	Sum
Min	

Report Title - your report title (default Value Over Time Report)

Saved Filter Project - select "Current" or Browse to find the project where you want to save this filter.

Saved Filter Name - the name of a saved filter to use when filtering collected data.

Use the Search tab to create the filter. After entering your search criteria, click **OK**.

If the preview displays the data you want, click **Save Filter** and provide a Project Name and Filter Name to specify where the filter is saved.

On the Run Report parameters page, select the filter. If you leave the Filter Name no filtering is performed.

Note: "Saved filters" can filter on start and end times. Because start and end times are specified in the Time Period parameter, there is a possibility you could select a combination of filter date and time periods that exclude all data. For example, you could select Time Period of "Yesterday" and a filter of Start=less than Jan 1, 1900. To avoid this problem, Electric Cloud recommends you do NOT include dates or times in your filter.

Table Column Heading Property - the table heading for the right column (default Count)

Table Column Heading Time - the table heading for the left column (default Date)

Table Time Grouping - these are: Auto (default), Minute, Hour, Day, Week, Month, Year

Time Period - these are keywords to define the report start and end times. The range is used to filter collected data. All values are in the local time zone.

Today - midnight until now

Yesterday - previous full day

This week - from the start of the previous Sunday until now

Last week - from Sunday to Sunday of the last complete week

This month - from the start of the first of this month until now

Last month - from the last complete calendar month, which means on January 3rd the last full month is all of December

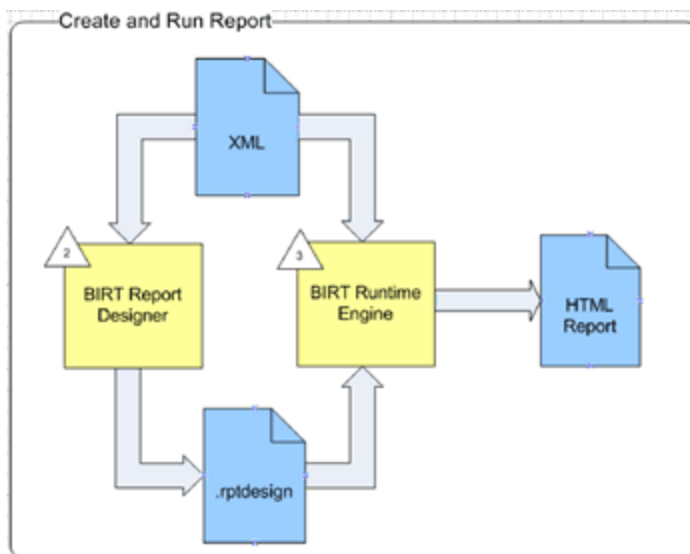
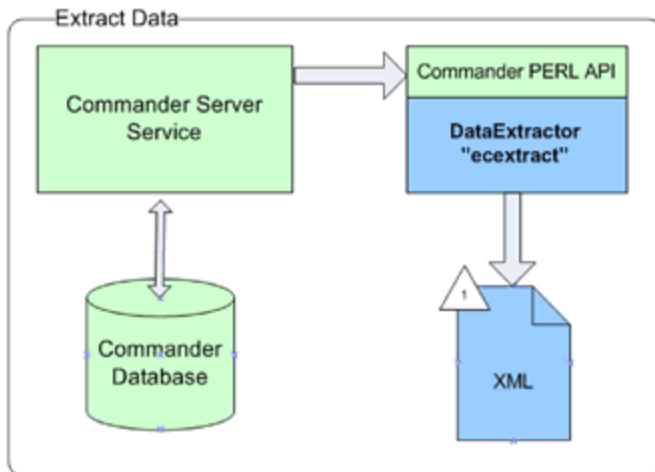
This year - from January 1st until now

Last year - all of last year (Jan 1 to December 31)

All - all dates until now

Creating Custom reports

The ElectricCommander batch mode reports which, by default, run once each night and are accessed through the Reports tab [in the Electric Cloud project] in the web UI. These reports are built using a combination of the Commander API and the BIRT Report Designer tool. BIRT is an open source Eclipse-based reporting system you can use to create your own custom reports, or modify existing Commander reports. This section describes how to write your own custom report and integrate it into ElectricCommander.



There are several steps to using BIRT to create reports:

- Create an XML file with data using the Commander Perl API
- Design a report using BIRT RCP Report Designer

- Run the report with the BIRT Report Engine
- View the report from within Commander

[Data extraction](#)

[Using `ecextract.pl` for data extraction](#)

[BIRT Report Designer](#)

[Understanding additional report components](#)

[Packaging reports for Commander](#)

[Helper functions in `ReportUtils.pm` That are Part of the ElectricCommander Perl Module](#)

Data extraction

To begin creating a custom report, you must first extract report data from Commander and move it into a standalone XML file. The recommended way to create the XML file is to use the Commander `ecextract` utility program, which is derived from the Perl API. The benefits of using `ecextract` to create XML files are:

- **Security** - `ecextract` enforces all ElectricCommander security and ACLs. (Direct DB access does not honor ACLs.)
- **Maintainability** - `ecextract` is supported by Electric Cloud, including backward compatibility and insulates you from database schema changes.
- **Simplicity** - `ecextract` provides easy-to-use search and filtering options. Because of the complex nature of our schema, particularly for properties, it may be difficult to create custom SQL statements.
- **Re-use** - You may find that XML files created by built-in reports already contain all the data you need. If you simply want to present data differently or "slice" the data in a different way, you can re-use the XML from the existing built-in reports.

Note: If you already use `ectool` or Perl for data extraction, and have already created the scripts you need for custom reports, you may continue to use those scripts. However, Electric Cloud recommends using `ecextract` if you are new to creating custom reports.

Using `ecextract.pl` for data extraction

A Perl program, `ecextract.pl`, is provided so you do not have to write Perl code—or you can use it as a starter for your own Perl API calls. This script allows you to specify the most common query types on a command line and generates an XML file. The following table lists currently available `ecextract` arguments:

Usage: `ecextract` [options]

Data Extract Options:	
<code>--type job jobStep resource</code>	The object type to query.

<code>--property [label=]property</code>	Add a property to the output. If label is specified, it is used as the XML tag name instead of the property name.
<code>--filter property, operator, operand1 [,operand2]</code>	Add a property filter to the query.
<code>--savedfilter filterName</code>	Use a saved filter (ignores <code>--filter</code>)
<code>--period period</code>	<p>The time period. One of: today, yesterday, thisweek, lastweek, thismonth, lastmonth, thisyear, lastyear, all, window, <number of days>, or range,<start-time>,<end-time></p> <p>Range start, end time is date in ISO 8601 format: YYYY-MM-DDTHH:MM:SS[Z]If the date ends with 'Z', it is considered to be GMT time, otherwise the local timezone is used.</p>
<code>--sort property, order</code>	Add a sort to the query. Order is either 'ascending' or 'descending'.
<code>--outputPath</code>	Path and name of output file. Default is <code>out.xml</code> .
<code>--max</code>	Maximum number of objects to return.
Server Communication Options:	
<code>--server server</code>	Address for the ElectricCommander server. Defaults to the value of the <code>COMMANDER_SERVER</code> environment variable. If it does not exist, defaults to localhost.

<code>--port port</code>	HTTP listener port on the server. Defaults to 8000.
<code>--securePort port</code>	HTTPS listener port on the server. Defaults to 8443.
<code>--secure 0 1</code>	If set to true, HTTPS is used to communicate with the server. Defaults to true.
<code>--timeout seconds</code>	Set the number of seconds for the timeout. Defaults to 280.
Login Options:	
<code>--user user</code>	ElectricCommander login user.
<code>--pass password</code>	Password for login user.
<code>--credentialName</code>	Credential name (valid only when running inside a job step).
Global Options:	
<code>--help</code>	Print this message and exit without doing anything.
<code>--version</code>	Debug level. Higher values... more details 0 - errors only 1 - 0 + System progress 2 - 1 + Property Warnings 3 and higher give detailed diagnostics
Plugin File Options:	
<code>--load loadFile</code>	Plugin perl program to execute
<code>--</code>	Stop processing ecextract options. Allows plugins to process their own options following this option.

ecextract.pl is part of the EC-Reports plugin—this script is located in the "agent/bin" directory of the EC-Reports plugin.

Notes:

--property

Properties can be specified as high-level values (--property propertyName), properties in a sheet (--property sheet1/sheet2/propertyName), or as relative paths (--property /myProject/propertyName).

--filter

Each entry consists of a property name to test and an operator to use for comparison. The property can be an intrinsic property defined by ElectricCommander or a custom property added by the user. Each operator takes zero or one operand to compare against the desired property.

The operators are:

contains, equals, greaterOrEqual, greaterThan, in, lessOrEqual, lessThan, like, notEqual, notLike, isNotNull, isNull

Example: --filter jobName,equals,Hello

Examples

The following 3 examples contain the script input and the output.

Example 1: Find properties on job steps for one day in April

```
ec-perl ecextract.pl ^
  --user report ^
  --pass report ^
  --server localhost ^
  --outputPath jobStepOut.xml ^
  --type jobStep ^
  --max 1000 ^
  --property jobId ^
  --property finish ^
  --property jobStepId ^
  --property stepName ^
  --property status ^
  --property outcome ^
  --property runTime ^
  --property waitTime ^
  --filter jobName,like,commander-2.2%% ^
  --filter createTime,greaterOrEqual,2008-04-01T00:00:00.000Z ^
  --filter createTime,lessOrEqual,2008-04-02T00:00:00.000Z ^
  --sort jobId,descending
```

Output from example 1

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<response>
  <object objectType="jobStep" objectId="jobStep-3511785">
    <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
    <finish name="finish">2009-04-01T23:54:45.424Z</finish>
    <jobStepId name="jobStepId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobStepId>
    <stepName name="stepName">Setup</stepName>
    <status name="status">completed</status>
    <outcome name="outcome">success</outcome>
    <runTime name="runTime">0</runTime>
    <waitTime name="waitTime">922</waitTime>
  </object>
  <object objectType="jobStep" objectId="jobStep-3511915">
    <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
    <finish name="finish">2009-04-01T23:59:26.844Z</finish>
    <jobStepId name="jobStepId">5da765dd-73f1-11e3-b67e-b0a420524153</jobStepId>
    <stepName name="stepName">Drop</stepName>
    <status name="status">completed</status>
    <outcome name="outcome">success</outcome>
    <runTime name="runTime">0</runTime>
    <waitTime name="waitTime">266</waitTime>
  </object>
  .....
</response>

```

Example 2: Find all jobs that were created in April 2008

```

ec-perl ecextract.pl ^
--user report ^
--pass report ^
--server localhost ^
--outputPath jobOut.xml ^
--type job ^
--max 100 ^
--property jobName ^
--property jobId ^
--property projectName ^
--property procedureName ^

```

```
--property branch ^
--filter createTime,greaterOrEqual,2009-04-01T00:00:00.000Z ^
--filter createTime,lessOrEqual,2009-04-30T00:00:00.000Z ^
--sort projectName,ascending ^
--sort procedureName,ascending ^
--sort createTime,descending
```

Output from example 2

```
<?xml version="1.0" encoding="utf-8"?>
<response>
  <object objectType="job" objectId="job-158461">
    <jobName name="jobName">commander-3.2.23916-200904011650</jobName>
    <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
    <projectName name="projectName">Commander</projectName>
    <procedureName name="procedureName">Master</procedureName>
    <branch name="branch">3.2</branch>
  </object>
  <object objectType="job" objectId="job-158455">
    <jobName name="jobName">commander-anders-main-ad.23915-200904011627</jobName>
    <jobId name="jobId">4fa765dd-73f1-11e3-b67e-b0a420524153</jobId>
    <projectName name="projectName">Commander</projectName>
    <procedureName name="procedureName">Master</procedureName>
    <branch name="branch">main</branch>
  </object>
  .....
</response>
```

Example 3: Find all resources

```
ec-perl ecextract.pl ^
--user report ^
--pass report ^
--server localhost^
--outputPath resourceOut.xml ^
--type resource ^
--max 100 ^
--property resourceName ^
```



```
--property stepCount ^
--property stepLimit ^
--property hostName ^
--filter stepCount,greaterThan,0 ^
--sort resourceName,ascending
```

Output from example 3

```
<?xml version="1.0" encoding="utf-8"?>
<response>
  <object objectType="resource" objectId="resource-106">
    <resourceName name="resourceName">ecbuild-lin1</resourceName>
    <stepCount name="stepCount">2</stepCount>
    <stepLimit name="stepLimit">0</stepLimit>
    <hostName name="hostName">ecbuild-lin1</hostName>
  </object>
  <object objectType="resource" objectId="resource-108">
    <resourceName name="resourceName">ecbuild-win1</resourceName>
    <stepCount name="stepCount">1</stepCount>
    <stepLimit name="stepLimit">0</stepLimit>
    <hostName name="hostName">ecbuild-win1</hostName>
  </object>
  .....
</response>
```

BIRT Report Designer

Now that you have a dataset, you can create a BIRT report. You need an XML file to use while designing the report. If you did not create an XML data file yet, now is the time to do so. After you have sample data, you are ready to start designing a report.

Report definitions are created with the BIRT Report Designer, currently version 2.5.2. These definitions are then stored in `.rptdesign` files. You have several options for installing the designer tool:

- Use a BIRT Report Designer bundled and already installed with Eclipse, or
- download BIRT Report Designer to use with your existing Eclipse installation, or
- because Eclipse is not required, you can download only the BIRT Report Designer (this is the easiest and preferred method).

Electric Cloud recommends downloading the BIRT RCP Report Designer only, which is the standalone design tool. The url for downloading the Report Designer is:

http://www.eclipse.org/downloads/download.php?file=/birt/downloads/drops/R-R1-2_5_2-201002221500/birt-rcp-report-designer-2_5_2.zip

The general download page for BIRT 2.5.2 with the other options discussed is

http://download.eclipse.org/birt/downloads/build.php?build=R-R1-2_5_2-201002221500

Using the BIRT Report Designer is beyond the scope of this help topic. Documentation and tutorials can be found on the BIRT website: <http://www.eclipse.org/birt>. You may find it helpful to view the report designs for built-in ElectricCommander reports. These designs can be found in the `agent/reports/<report name>` directory in the EC-Reports plugin.

Understanding additional report components

In addition to the XML dataset and report design, there are optional report components you may wish to include in your report.

- **External style sheet**
BIRT supports external style sheets for reports. If you use these style sheets, they need to be located with your report design. For example, built-in ElectricCommander reports use an external style sheet file `"styles_birt.css"`.
- **Externalized string properties**
String literals (used in your report design) can be externalized into property files to support localization. These property files need to be located with your report design also. The properties file for the default locale will have the same name as your report and a `".properties"` file extension, for example, `CategoryPie.properties`. Text for additional locales are stored in files named `<report name>_<locale>.properties`, for example, `CategoryPie_ja.properties` contains text localized for the Japanese language.

You may review examples of these additional components for built-in Commander reports. These files are located in the EC-Reports plugin, in the `agent/reports/<report name>` directory.

Packaging reports for Commander

After you have an XML dataset (using `ecextract.pl`), a BIRT report design (using BIRT Report Designer), and any additional optional report components completed, you are ready to integrate your report into Commander. To do this:

- Store all the report components in a location accessible to your Commander agent—you can use the Commander plugins directory. For example, for your report files, you could create a directory called `"MyReport"` in the Commander plugins directory.

- Create a Commander procedure to perform the necessary steps to generate a report, including: extracting data, running the report design, and registering report output to appear on the Report tab on the Project Details page. You may want to copy one of the procedures from the built-in Commander reports to use as a template. The following command creates a copy of the RunReport_CategoryPie procedure called RunReport_MyReport in the project of your choice. Substitute your project name for <my-project-name> in the following command.

```
ectool clone --cloneName/projects/<my-project-name>/procedures/RunReport_
MyReport
--projectName/plugins/EC-Reports/project --procedureName RunReport_
CategoryPie
```

Example: customizing the command body

The following is an example of customizing the procedure contents for the RunReport_MyReport procedure, cloned from the RunReport_CategoryPie procedure above.

One customization is required and two are optional. The remainder of the command body does not need to be changed.

- Customizing the location of the BIRT report design file
Required: update the location of the BIRT report design file.
 - **Original value:** our \$reportDesignFile =
"\$pluginDir/agent/reports/CategoryPie/CategoryPie.rptdesign";
 - **New value:** our \$reportDesignFile = "\$ENV{'COMMANDER_PLUGINS'}
/MyReport/MyReport.rptdesign";
- Customizing data extraction
To customize data extraction using `ecextract.pl`, modify the following section of the command body:

```
#-----
# extractReportData
#
#       Extract report data from Commander using ecextract utility
#-----

sub extractReportData()
{
    # Time property varies based on object type
    my $timeprop = "modifyTime";
    if (
        $timeprop = "finish";
    )
    my $command = "ec-perl \"$pluginDir/agent/bin/ecextract.pl\" "
        . " --debug 3"
        . " --outputPath \"$ ecextractXmlFile\""
        . " --credential \"$[Credential]\""
}
```

```
        . " --type \"${Object Type}\"
        . " --property time=$timeprop"
        . " --property series1=\"${Property}\";
    if ("${Saved Filter}" ne "") {
        command = $command . " --savedFilter \"${Saved Filter}\"
    }

    $command = $command . " --period \"${Time Period}\";
    print "data extract [$command]\n";
    my $result = ` $command `;
    print "extract result:\n$result";
    if ($?) {
        exit (1);
    }
}
```

- Customizing report parameters

To customize the parameters passed to your report design, modify the following section of the command body:

```
#-----
# getReportArgs
#
#      Get report arguments
#-----

sub getReportArgs() {
    # args for creating report
    my $categoryHeading = ucfirst("${Property}");
    my @args = (
        "-f", "$reportFormat",
        "-o", "$reportName",
        "-i", "images/",
        "-p", " xmlfile=../$extractXmlFile",
        "-p", "ReportTitle=$reportTitle",
        "-p", "TableColHeadingTime=$categoryHeading",
        "-p", "TableColHeadingSeries1=Count" ,
        "-p", " SearchURL=" . getSearchUrl(),
        "$reportDesignFile",
    )
}
```

```

);

# add locale args
foreach my $locale (@locales) {
    if ($locale ne "") {
        unshift @args, "-l $locale";
    }
}

return @args;
}

```

Helper functions in ReportUtils.pm That are Part of the ElectricCommander Perl Module

ecgenReport - runs the BIRT report, using the BIRT Report Designer

- **installationDirectory** - this is the directory where ElectricCommander is installed and where you will find JAVA and BIRT
- **artifactDirectory** - this is the directory where reports, images, and other artifacts should be stored. To display using the Reports subtab, these items must be in a directory named "artifacts" at the top of the job's workspace
- **args** - an array of arguments to the BIRT runtime engine, which specifies parameters, design templates, and so on...

registerReport - marks the report so it will be displayed when you select the Reports subtab

- **commander** - an `ElectricCommander()` object used to make Commander calls. If you need to use a specific user context, call the login method before calling `registerReport`
- **jobId** - this is the report job ID, which is used to set properties on the job
- **url** - this is the URL to register and it takes the form workspace name/html file name
- **title** - this is the text string for the report navigation tree
- **treeGroupings** - a hash grouped `name/values` set as properties to define the report tree groupings—report tree groupings are defined on a propertySheet named `ec_reportConfigurations` on the project or on the server
 - **hash key** - this is a property name that will be created on job's `ec_reportData/$title/` property sheet and used for report tree grouping
 - **hash value** - property value

registerArtifactsDirectory - creates the property required to set the artifacts directory

- **commander** - an `ElectricCommander()` object used to make Commander calls
- **jobId** - this is the report job ID, which is used to set properties on the job

- **artifactsDir** - this is a directory path relative to the workspace where job artifacts will be stored

extractFile - this function pulls content from the report definition (stored in properties) and creates files from them

- **commander** - an `ElectricCommander()` object used to make Commander calls. If you need to use a specific user context, make sure to call the `login` method before calling `registerReport`
- **jobId** - this is the report job ID, which is used to set properties on the job
- **reportType** - this is used to lookup report attributes
- **files** - an array of hashes that specify which properties should be loaded into which files
 - **prop** - the property name from the installed report type
 - **dest** - the destination file name (relative to the current working directory)
 - **expand** - this is "1" if you want Commander to expand property references in the property, "0" if not

getReportTimestamp - return hires timestamp formatted in a specified time zone

- **TimeZone** - (optional) a time zone specifier, default is "local"
- **FormatString** - (optional) a format specifier for the timestamp string

Custom Report Examples

This help topic contains two examples illustrating how you might use the BIRT Report Designer to modify a built-in Commander report to create a custom report more meaningful for your purposes.

Example 1: modifying an existing report - adding a "banner" heading

The easiest way to customize a report is to take an existing report and modify it. Commander provides a set of built-in reports you can modify.

Process overview for modifying a Commander built-in report

- Make a copy of an existing report
- Make changes to the report
- Test your new custom report

To copy an existing Commander report

For this example, we will make a copy of the built-in Value Over Time report. The process to copy a report is:

- Make a copy of the file system report components of the Value Over Time report located in the ElectricCommander plugins directory.

Rename the copied directory to MyReport.

The default location of the ElectricCommander plugins directory is:

On Linux: `/opt/electriccloud/electriccommander/plugins`

On Windows: `C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\plugins`

Linux example - The following commands will make a copy of the Value Over Time report file components and then rename the files:

```
$ cd /opt/electriccloud/electriccommander/plugins
$ cp -r EC-Reports-1.0.0.*/agent/reports/ValueOverTime/ MyReport
$ cd MyReport
$ rename ValueOverTime MyReport ValueOverTime*
```

The contents of the MyReport directory should be:

```
$ ls -l
MyReport_en.properties
MyReport_ja.properties
MyReport_ko.properties
MyReport.properties
```

```
MyReport.rptdesign
```

```
MyReport_zh.properties
```

```
styles_birt.css
```

- Make a copy of the RunReport_ValueOverTime procedure from the EC-Reports plugin that performs the steps to generate a report, including extracting data, running the report design, and registering the report output to appear on the Report tab of the Project Details page. In this example, we will create a Project called “MyReportProject” and the procedure we copy will be called “RunReport_MyReport”.

```
$ ectool createProject MyReportProject
```

```
$ ectool clone --cloneName /projects/MyReportProject /procedures/RunReport_
MyReport
```

```
--projectName /plugins/EC-Reports/project --procedureName RunReport_
ValueOverTime
```

```
response requestId="1"
```

```
<cloneName>RunReport_MyReport</cloneName>
```

```
</response>
```

- Edit the “RunReport_MyReport” procedure in “MyReportProject” to reference the report design file copied earlier.
- Navigate to project MyReportProject, procedure RunReport_MyReport and select the RunCustomReport step to edit the Command block:

Procedure Details – RunReport_MyReport Run Edit Access Control

Procedure Steps						Create Step
Step Name	Resource	Action	Parallel	Time Limit	Actions	
RunCustomReport		\$ =1; use utf8; use ElectricCommander; ...			Copy Delete	

Find the following line:

```
our $reportDesignFile =
"$pluginDir/agent/reports/ValueOverTime/ValueOverTime.rptdesign";
```

and change this line to:

```
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/ MyReport/MyReport.rptdesign";
```


Edit Step – RunCustomReport Access Control

General

Name:

Description:

Command

Command(s):

```
our $reportType = "[Report Type]";
if ($reportType eq "") { $reportType = $reportTitle; }
our $reportFormat = "html"; #html,pdf,doc,xls
our $reportName = safeFileName("$reportTitle.$reportFormat");
our $artifactsDir = "artifacts";
our $agentInstallDir = getAgentInstallDir();
our $pluginDir = "$ENV{'COMMANDER_PLUGINS'}[/plugins/EC-Reports/pluginName]";
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/MyReport/MyReport.rptdesign";
our $birtInstallDir = "$ENV{'COMMANDER_PLUGINS'}/[/plugins/EC-ReportEngine]/agent";
our $extractXmlFile = safeFileName("$extract-$reportType-$[/increment /myJob/xmlCounter].xml");
our @locales = getLocales();
```

Click **OK** to save the change.

To modify the copied report design

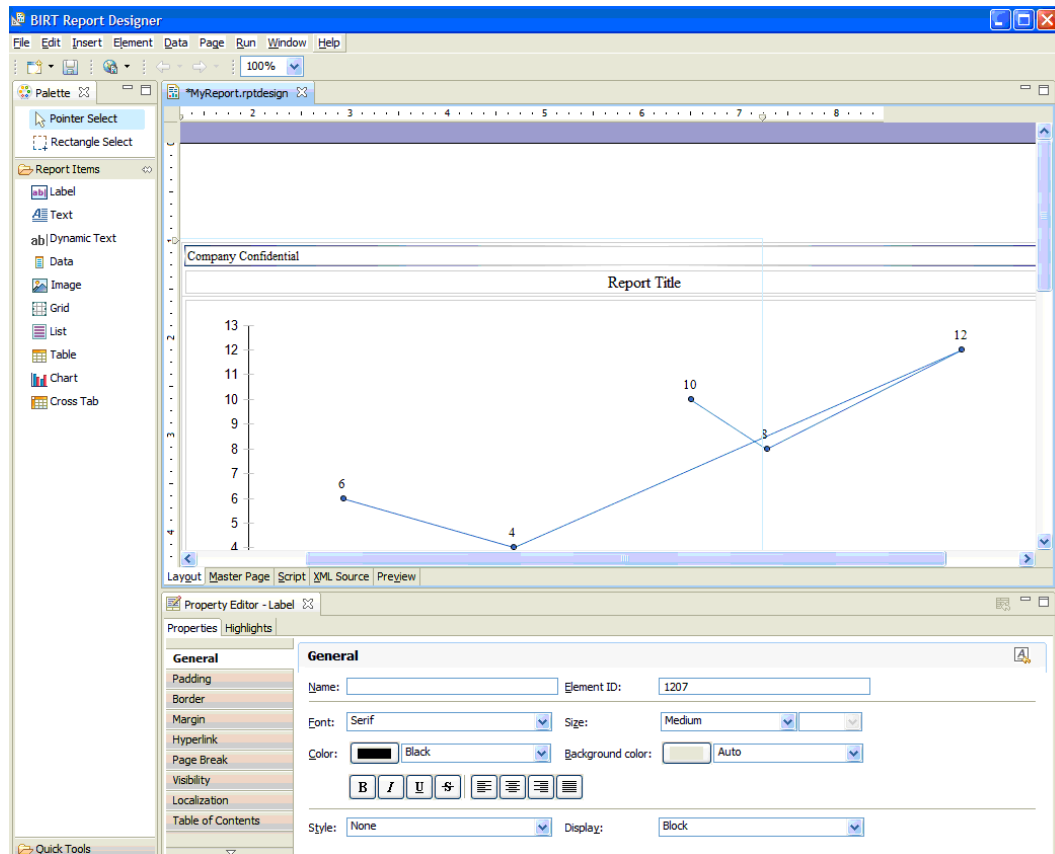
Start the BIRT Report Designer and open the report design file (`MyReport.rptdesign`) located in the `MyReport` directory previously created in the ElectricCommander plugins directory.

At this point... You are now ready to begin editing an existing report to create a custom report.

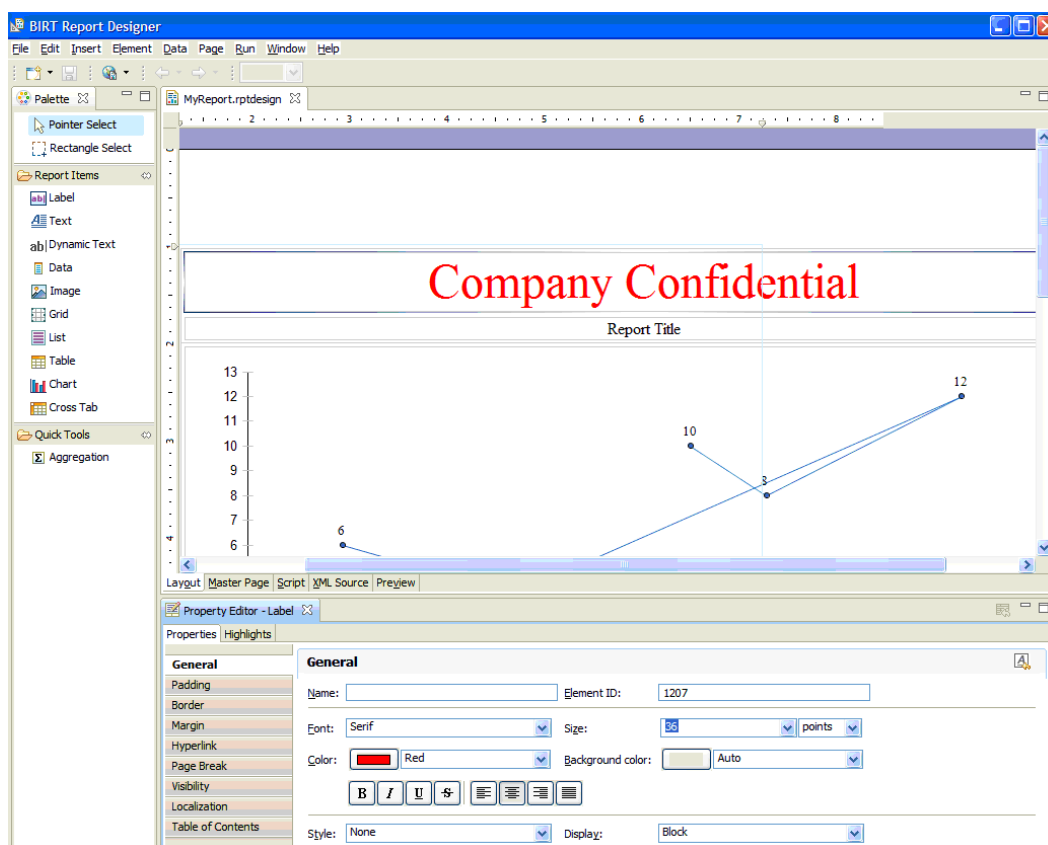
For our example, we are adding a new heading, "Company Confidential," (in large red letters) to the top of the existing report.

1. On the BIRT screen, select the Layout tab.
2. Drag a "Label" from the Palette window into the top of the report and type "Company Confidential" (or whatever text you would like to use for your test report) into the Label box.

See the next BIRT screen example.



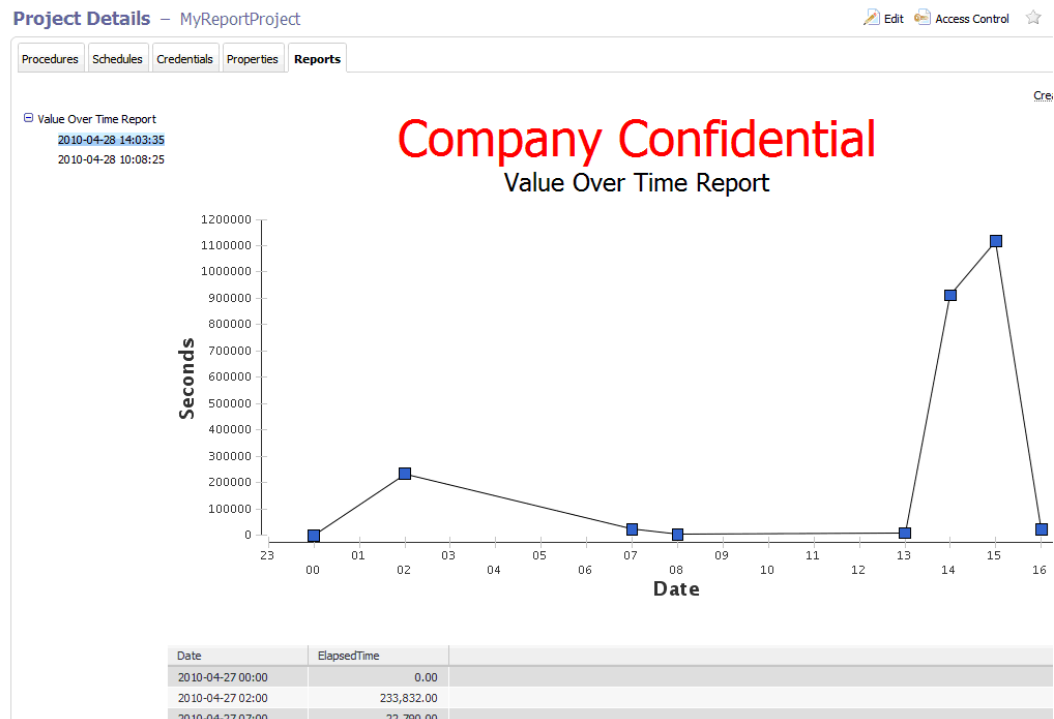
3. Now you can change the font size and color to create a banner appearance. Select the Property Editor window to customize your text - see the next screen example.
4. After customizing your text, Save your report.



Congratulations! You have just created a custom report.

Test your new report

Run the report by running the RunReport_MyReport procedure in the MyReportProject project. When the job completes, your new report will be available in the Links section on the Job Details page and from the Reports tab if the MyReportProject project. You will see the new header in the report output.



Example 2: complete end-to-end example

Using BIRT and some scripting expertise, you can create almost any kind of report you can imagine. This example creates a new report that shows the runtime trend of all job steps. Tasks for creating this report can be grouped into the following steps:

- Plan your report
- Create a new BIRT rptdesign file
- Deploy your custom report
- Test your new report

Planning this report example

We decided our report will be a single-series type report that will show job steps run time. The report will include a line chart at the top of the page whose X and Y axis represent the number of seconds and the job step ID respectively. We also want a table with two columns—column names will be "Job Step Name" and "Run Time".

Next, we need to gather all materials:

Generate sample data to test in the report.

You can use data from a previous run of a Value Over Time report—this file is located in the workspace of a previous reporting job and has the name ecextract.xml.

Also from the copy of a Commander Value Over Time report, we will "copy and paste" a few values from it to save some typing.

Note: This example briefly describes items we are working with, but for more information about BIRT, you may need to consult BIRT documentation.

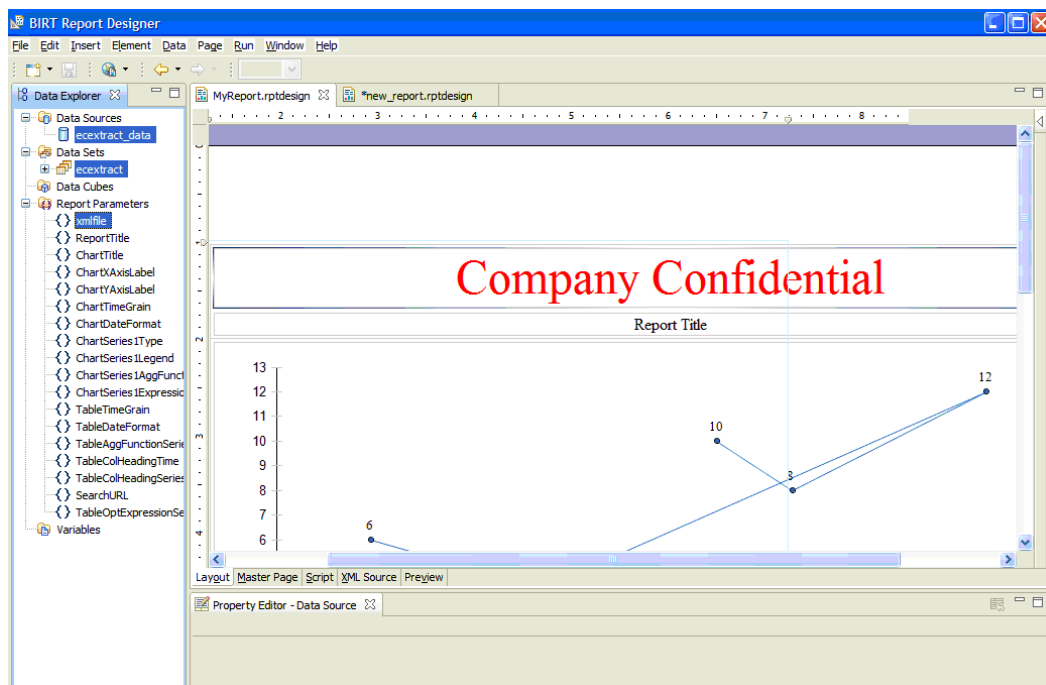
Creating a new BIRT rptdesign file

In BIRT Report Designer, create a new rptdesign file. Open the report file you created in "Example 1: modifying an existing report - adding a banner heading" so you can use the previous example report to copy some items into your new report.

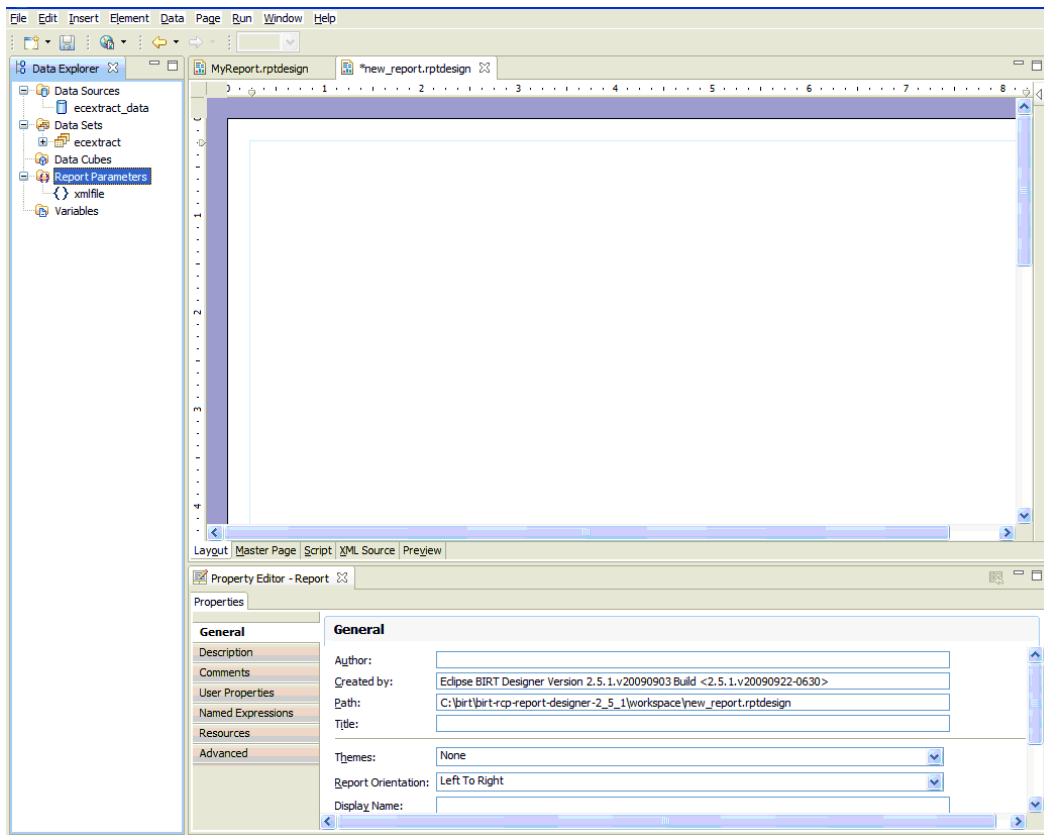
Items to copy into the new report:

- From Data Sources, copy ecextract_data
- From Data Sets, copy ecextract
- From Report Parameters, copy xmlfile

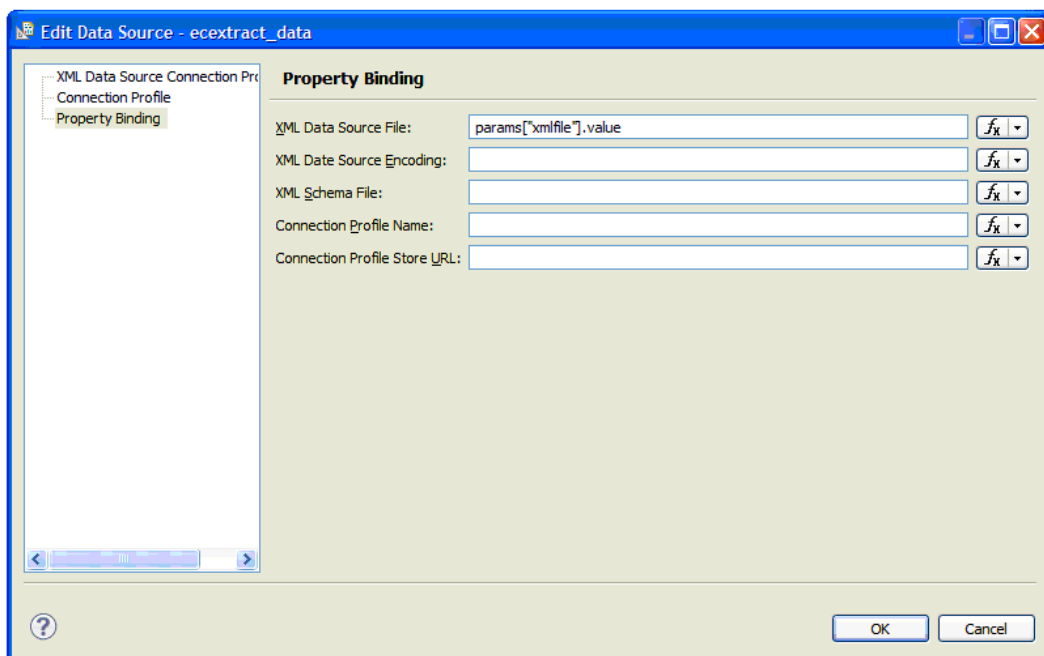
See the next screen example to help find these items.



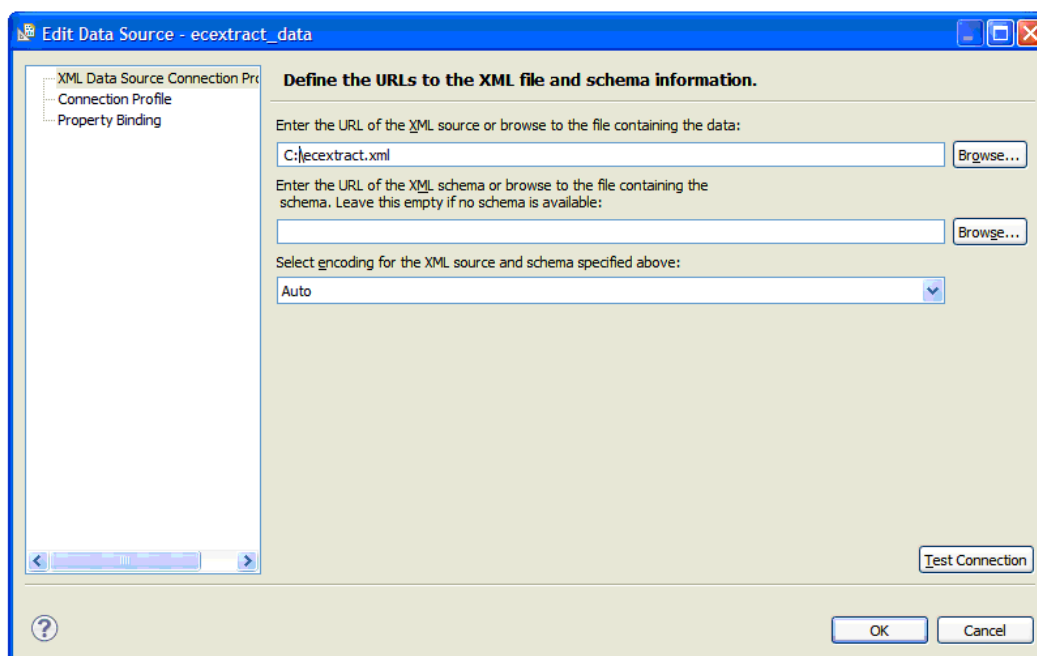
After copying these items, your new report design will look like the following example:



Double-click the Data Sources > ecextract_data file to ensure the XML file is referenced as a parameter. If it is not, type in the value as shown below.

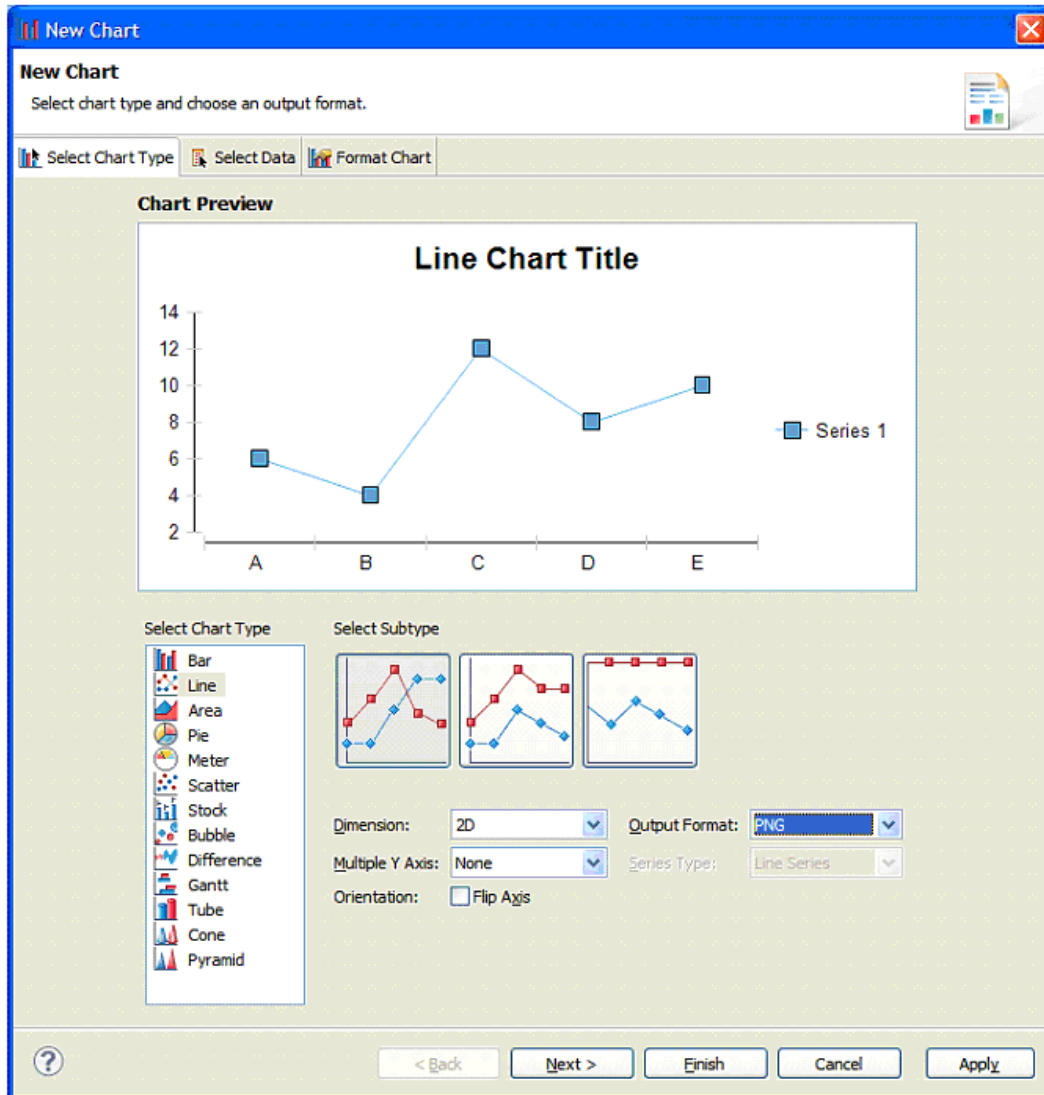


Double-click Data Sources > `ecextract_data` to ensure the XML Data Source Connection refers to the data file you downloaded earlier. In this example the file is `C:\ecextract.xml`.



From the Palette window, drag a Chart object into your report. Your view will be similar to the following:

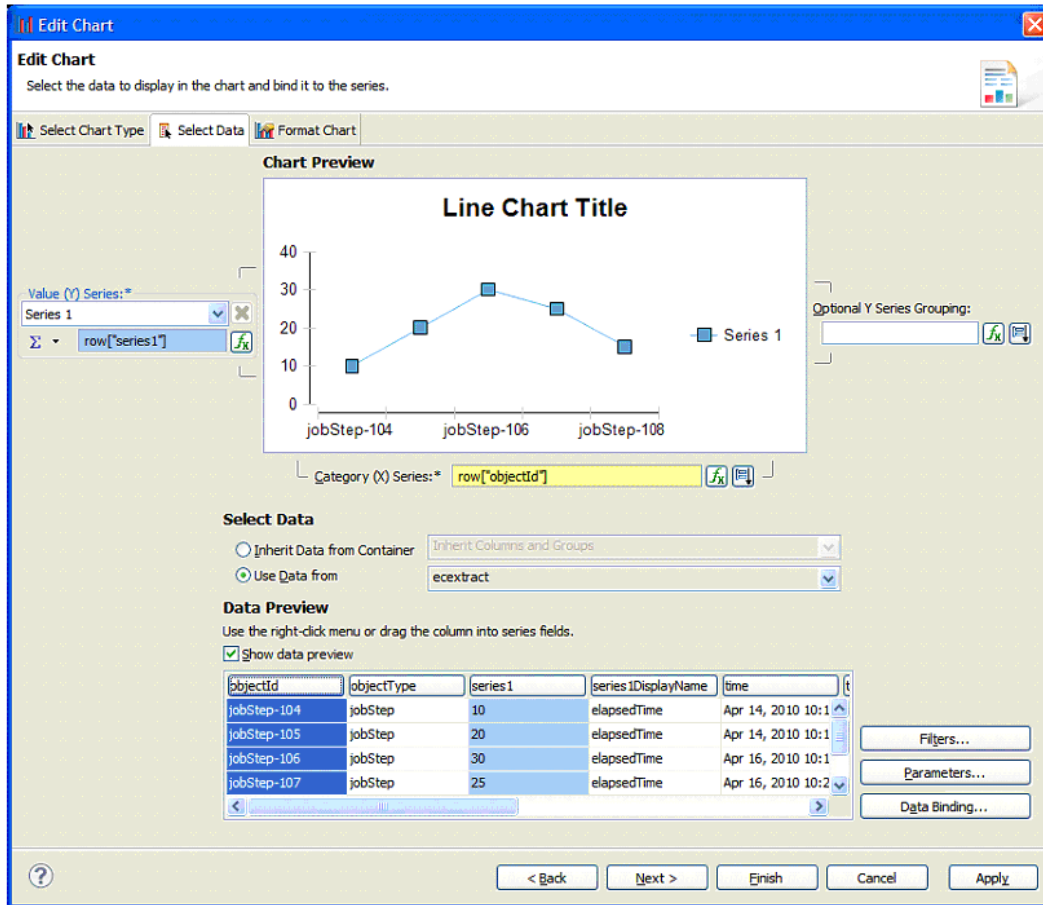
In the New Chart dialog, select Line under Select Chart Type and browse to select PNG for the Output Format.



Click **Next**.

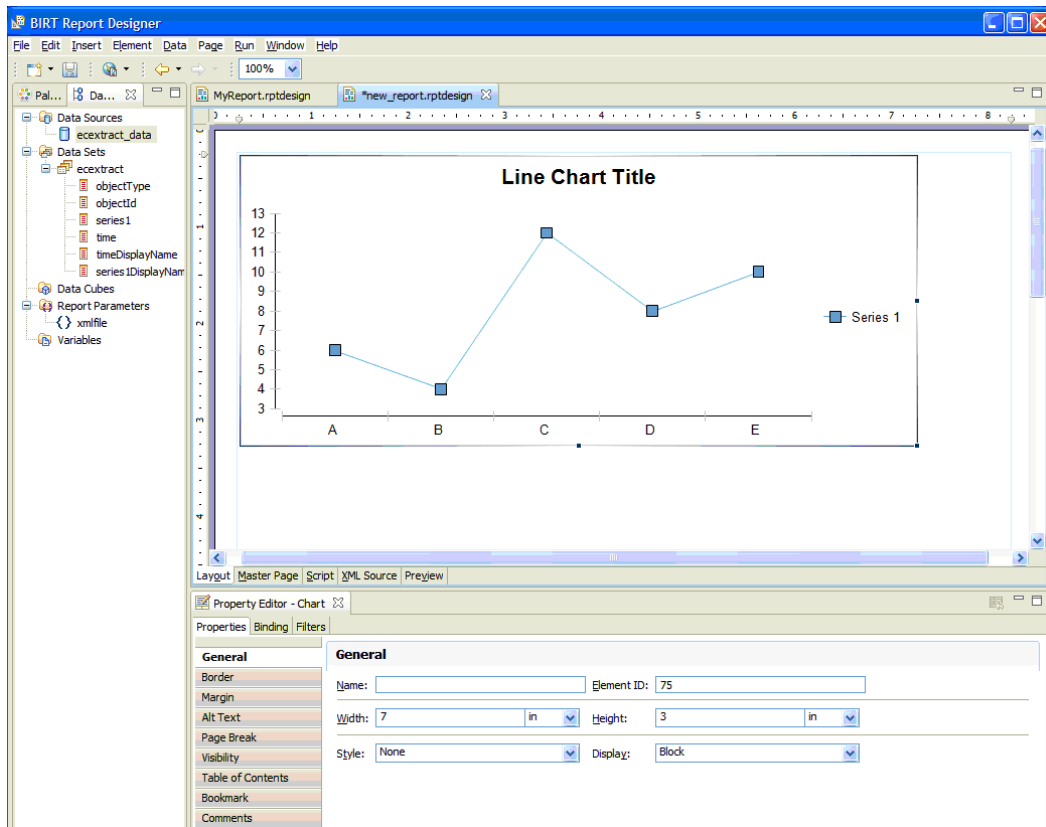
On the next screen:

- Under Select Data, browse to select ecextract.
- Make sure the Use Data From radio button is selected.
- Drag the "series1" header to the Value (Y) Series field.
- Drag the "objectId" header to the Category (X) Series field.



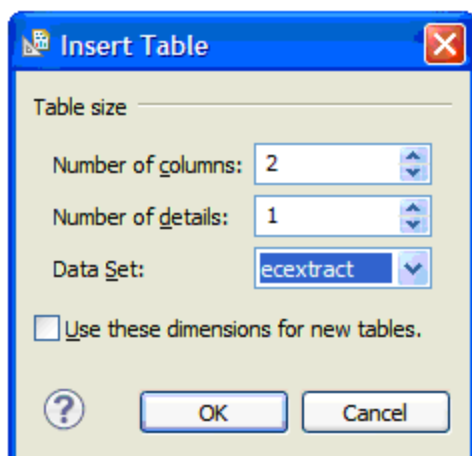
Click **Next** to apply any other formatting changes you choose, then click **Finish**.

On the next screen, after clicking Finish, you have to option to resize the chart as you wish.

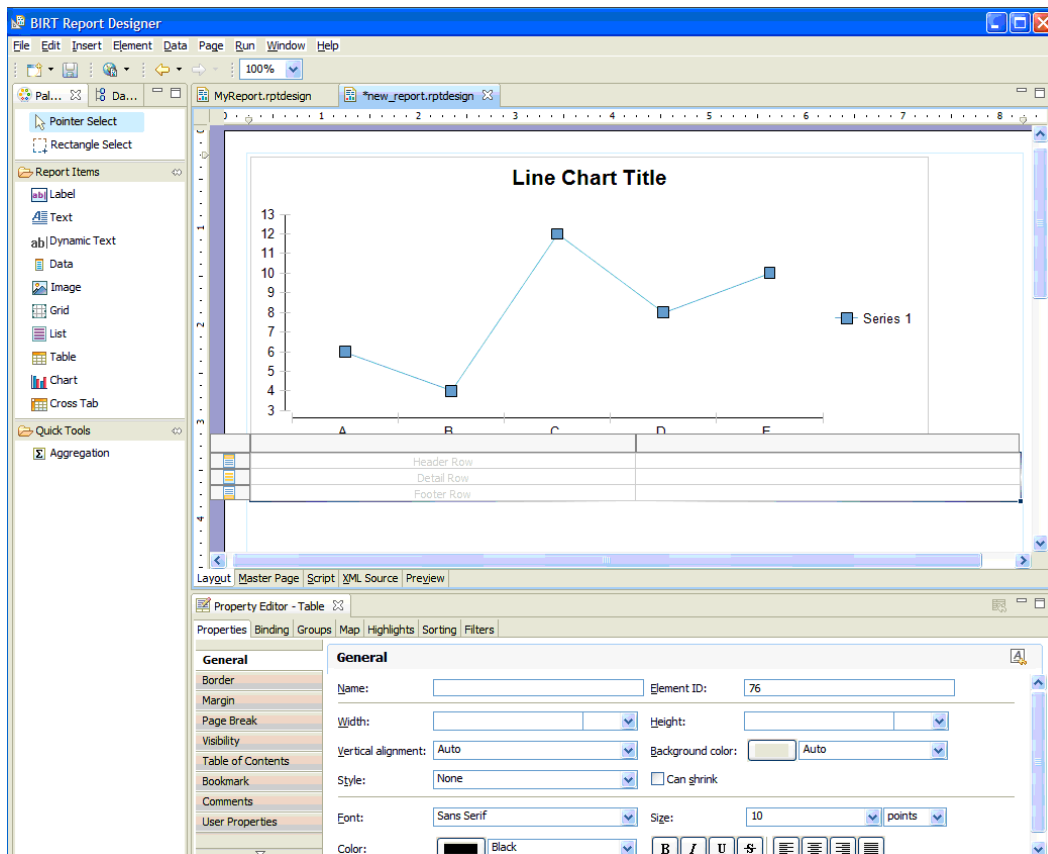


You are now ready to drag a Table item from the Palette to the report layout below the chart.

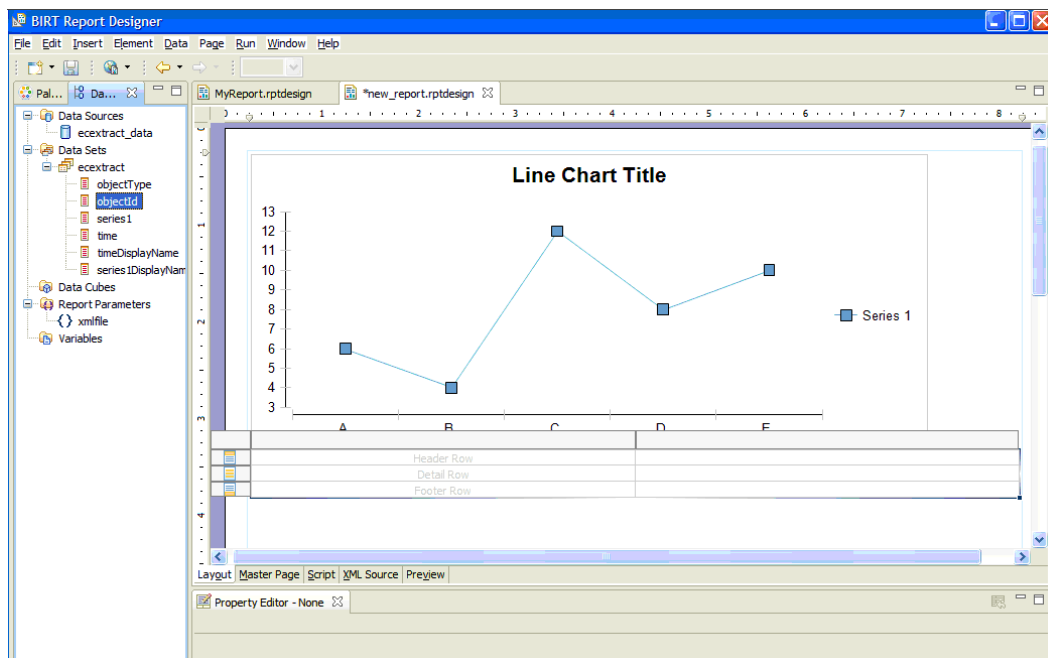
Select "2" columns and the ecextract dataset.



Click **OK**.



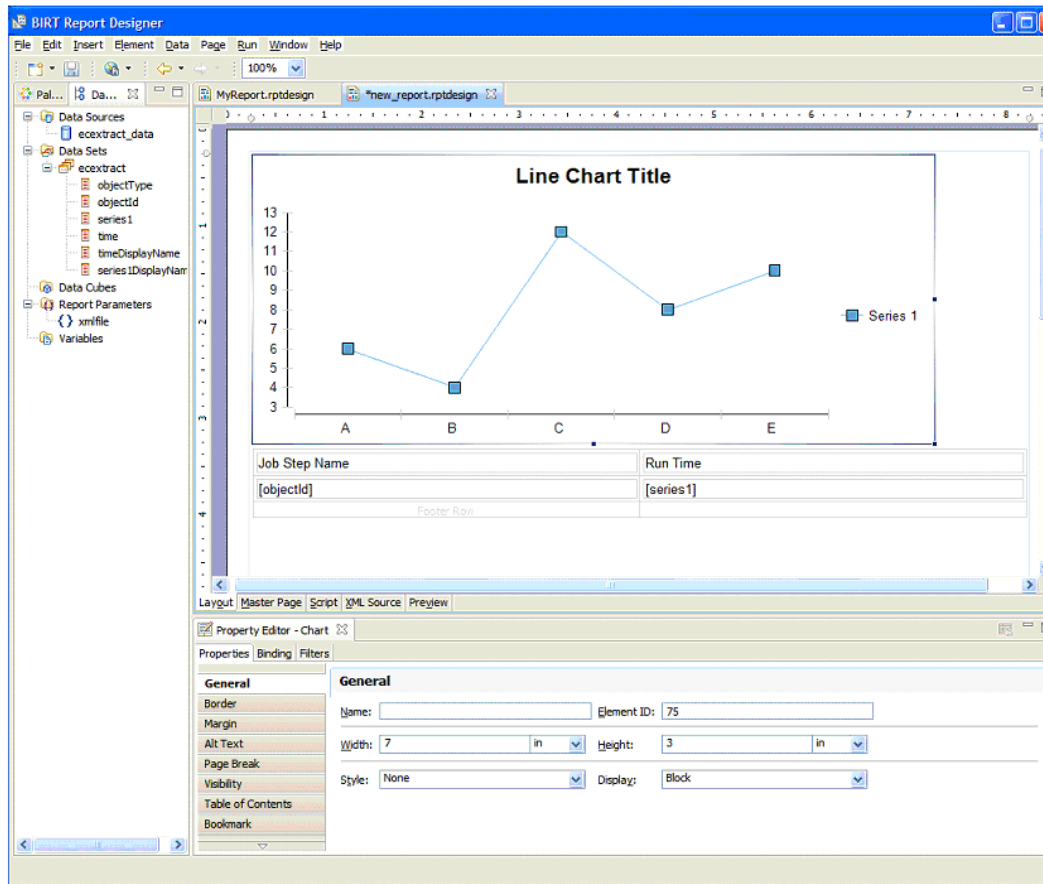
Select the Data Explorer tab and expand Data Sets > ecextract. See the next screen.



Now you are ready to:

- Drag "objectId" to the left Detail Row.
- Drag "series1" to the right Detail Row.
- Replace the Header Row for "objectId" and "series1" with Job Step Name and Run Time, respectively.

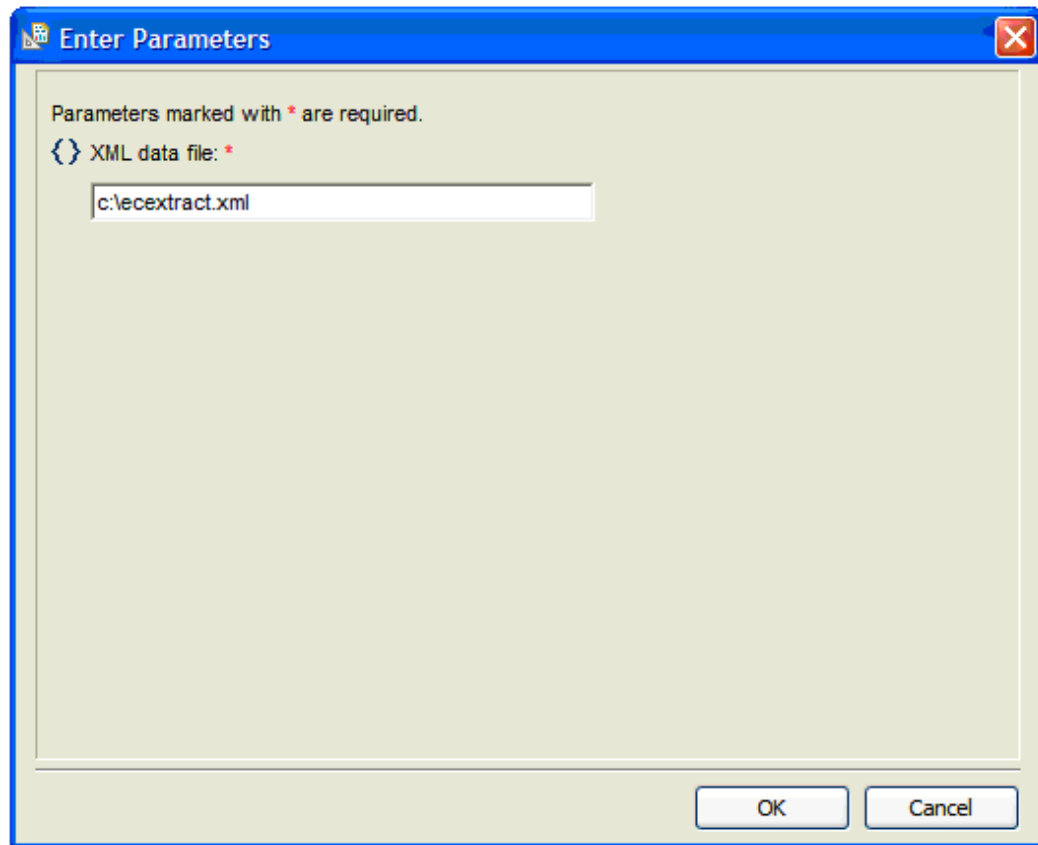
See the next screen example.

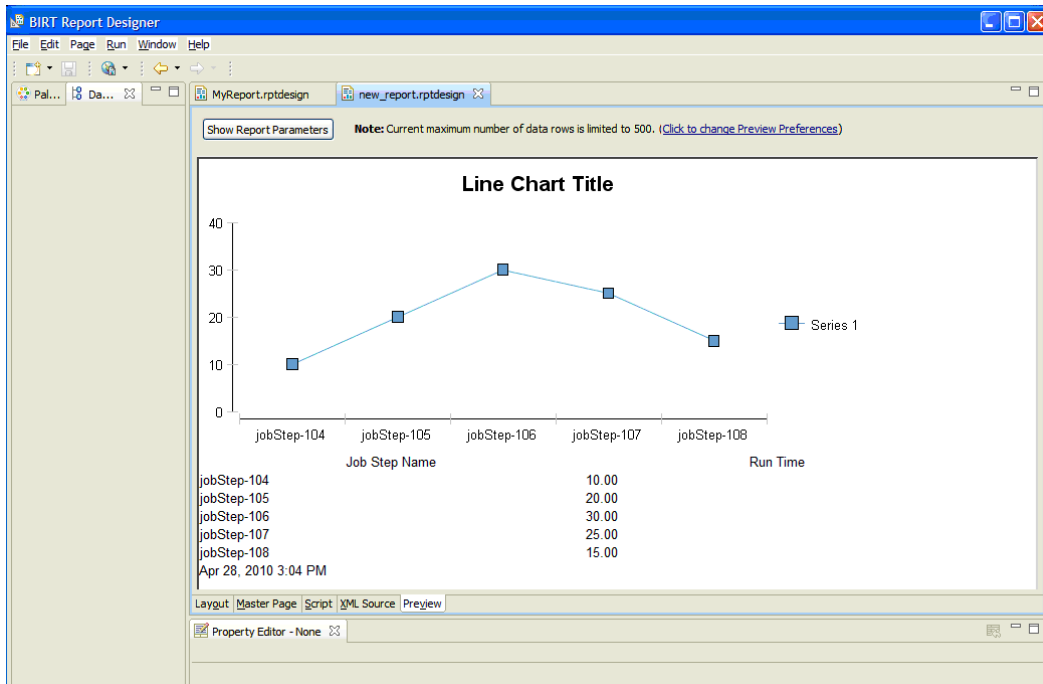


Select Preview to see your report.

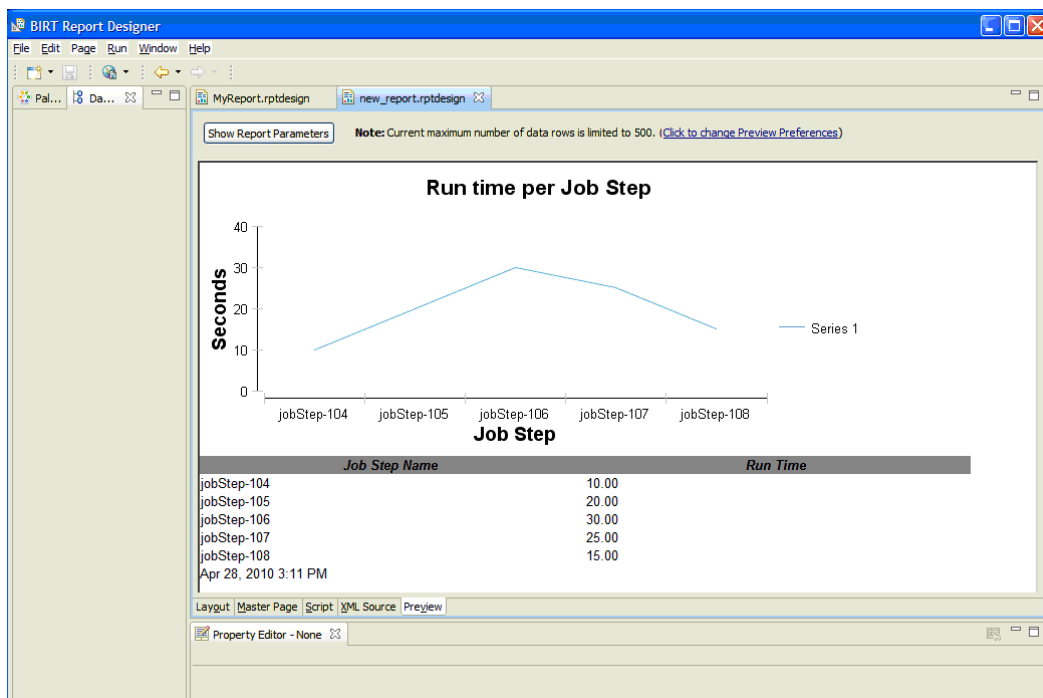
In the "Enter Parameters" dialog, type the location of your ecextract.xml data file.

Click **OK**.





At this point, you can continue to make formatting changes to make your custom report look exactly how you want it to look. Our example report, when finished, looked like the following example::



Deploy your custom report

If you have not already done so, save your new report design now. We have named our report design "JobStepRunTime.rptdesign".

The new report design needs to be stored in a location accessible to the Commander Agent that will run the report. As in the previous example, we will use the ElectricCommander plugins directory for this purpose.

- Save the JobStepRunTime.rptdesign file to a new JobStepRunTimeReport directory in the ElectricCommander plugins directory.

The default location of the ElectricCommander plugins directory is:

On Linux: /opt/electriccloud/electriccommander/plugins

On Windows: C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander\plugins

Linux example - The following commands will make a copy of the Value Over Time report file components and then rename the files:

```
$ cd /opt/electriccloud/electriccommander/plugins
$ mkdir JobStepRunTimeReport
```

- Now copy the previously saved JobStepRunTime.rptdesign file to the JobStepRunTimeReport directory.
- Make a copy of the RunReport_ValueOverTime procedure from the EC-Reports plugin that performs the steps to generate a report, including extracting data, running the report design, and registering the report output to appear on the Report tab of the Project Details page. In this example, we will create a Project called "JobStepRunTime" and the procedure we copy will be called "RunReport_JobStepRunTime".

```
$ ectool createProject JobStepRunTime

$ ectool clone --cloneName /projects/JobStepRunTime/procedures/RunReport_
JobStepRunTime

--projectName /plugins/EC-Reports/project --procedureName RunReport_
ValueOverTime

<response requestId="1"

    <cloneName>RunReport_JobStepRunTime</cloneName>

</response>
```

- Edit the "RunReport_JobStepRunTime" procedure in project "JobStepRunTime" to reference the report design file copied earlier.
- Navigate to project JobStepRunTime, procedure RunReport_JobStepRunTime and select the RunCustomReport step to edit the Command block:

Procedure Details – RunReport_JobStepRunTime

Run Edit Access Control

Create Step

Step Name	Resource	Action	Parallel	Time Limit	Actions
RunCustomReport		<pre>\$ =1; use utf8; use ElectricCommander; ...</pre>			Copy Delete

Find the following line:

```
our $reportDesignFile =  
"$pluginDir/agent/reports/ValueOverTime/ValueOverTime.rptdesign";
```

and change it to:

```
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/  
JobStepRunTime/JobStepRunTime.rptdesign";
```

Edit Step – RunCustomReport

General

Name:

Description:

Command

Command(s):

```
our $reportType = "${Report Type}";  
if ($reportType eq "") { $reportType = $reportTitle; }  
our $reportFormat = "html"; #html,pdf,doc,xls  
our $reportName = safeFileName("$reportTitle.$reportFormat");  
our $artifactsDir = "artifacts";  
our $agentInstallDir = getAgentInstallDir();  
our $pluginDir = "$ENV{'COMMANDER_PLUGINS'}/$[/plugins/EC-Reports/pluginName]";  
our $reportDesignFile = "$ENV{'COMMANDER_PLUGINS'}/JobStepRunTime/JobStepRunTime.rptdesign";  
our $birtInstallDir = "$ENV{'COMMANDER_PLUGINS'}/$[/plugins/EC-ReportEngine]/agent";  
our $ecextractXmlFile = safeFileName("$ecextract-$reportType-$[/increment /myJob/xmlCounter].xml");  
our @locales = getLocales();
```

Click **OK** to save the change.

Test your new report

Run the report by running the RunReport_JobStepRunTime procedure in the JobStepRunTime project. On the Run Procedure page, change the values of the following parameters:

- Object Type => Job Step
- Property Expression => /1000
- Report Title => Job Step Run Time

Run Procedure – RunReport_JobStepRunTime

Parameters

Chart Time Grouping:

Chart Type:

Chart X-Axis Label:

Chart Y-Axis Label:

Credential: Username: Password:

Locales:

Object Type:

Property:

Property Expression:

Property Function:

Report Title:

Report Type:

Resource:

Saved Filter Project: ☒ Current ☐ Browse

Saved Filter Name: Browse

Table Column Heading Property:

Table Column Heading Time:

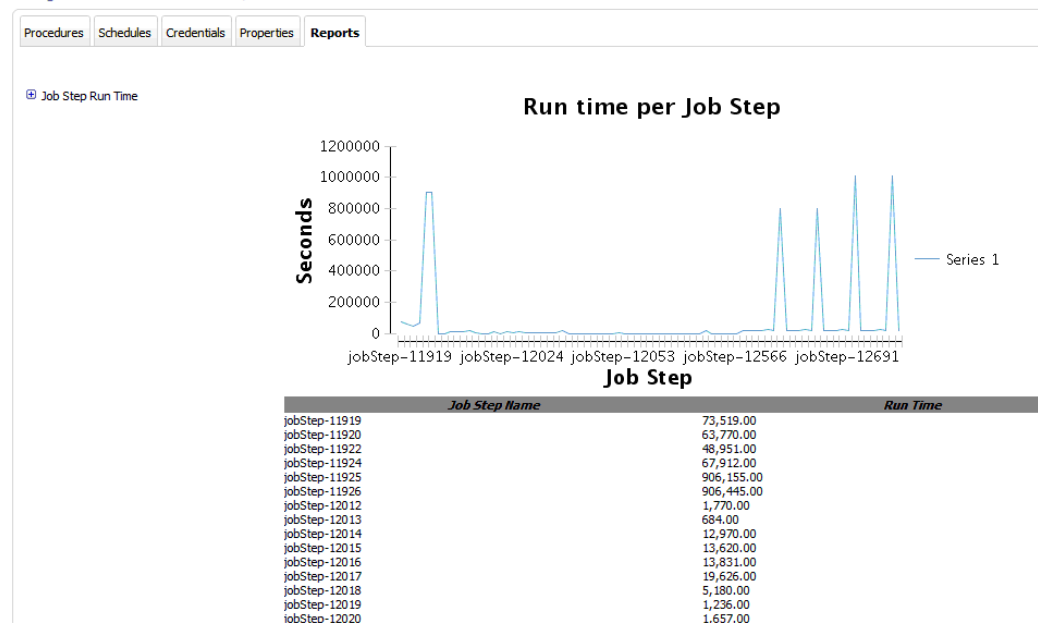
Table Time Grouping:

Time Period:

Click **Run**.

When the job completes, your new report will be available in the Links section of the Job Details page and from the Reports tab in the JobStepRunTime project.

Project Details – JobStepRunTime



Workflow Overview

[Workflow objects](#)

[Access Control](#)

[Property Search Paths](#)

[Parameters](#)

[Sending Notifications](#)

[Workflow logs](#)

[A Workflow Tutorial](#)

Use a Workflow to design and manage processes at a higher level than individual jobs. Workflows allow you to combine procedures into processes to create build-test-deploy lifecycles (for example). A workflow contains *states* and *transitions* you define to provide complete control over your workflow process. The Commander Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs.

Some benefits of defining a workflow include

- **Manual intervention** - workflows make it easy to introduce manual decision points into your process. With workflows, you can enable specific users or groups to choose how the workflow should proceed. Using an email notification, you can notify one or more users about the workflow status. If the workflow is waiting for someone to manually intervene, an email (including a screen link) can provide immediate attention and access to choices for how the workflow should proceed.
- **Aggregation** - workflows allow you to collect information from a set of related jobs into a single location. A workflow can act as a central place to monitor the full lifecycle of your build, test, and deploy process.
- **Looping** - workflows provide the native ability to run a job more than once. You have full control over the loop condition—you can re-run the job any number of times, only re-run the job if it did not succeed, or make this decision based on a custom job property.
- **Branching** - workflows allow you to run multiple jobs in parallel and wait for them to complete before moving to the next state in the workflow. If you combine branching with looping, you can re-run these jobs based on a condition you defined (for example, only re-run jobs that failed).
- **Multipathed** - design a workflow to take multiple paths automatically, depending on the outcome of each state's action, and the pre- and post-triggers you defined for each state.

Basic workflow concepts

Workflows contain states—one or more states can be designated as "starting" states to provide multiple entry points into the workflow. When a workflow is launched, a starting state is specified. You may prefer to design a complex or multi-purpose workflow so you can start the workflow at a different state to complete only a portion of the workflow for a particular outcome. For example, you might choose to run only the test suite workflow section.

As each state in a workflow becomes active, it performs an *action*. The state's action can be to create a job from a procedure or to start a workflow from a workflow definition. When the job or called workflow completes, "On Completion" transitions are evaluated to see if the workflow should change to a different state, possibly based on the outcome of the action.

You can create as many state definitions as you need. For example, you might have a small workflow definition with only 5 states, calling 5 procedures, or you might create 50-100 or more states to process a like-number of procedures from multiple projects.

A state can send email notifications before or after an action to notify interested users about workflow progress or to request manual user intervention to move the workflow to the next state.

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 – evaluates before sending email notifiers or starting the action

On Start – evaluates immediately after starting the action. These transitions are ignored if no action is specified for the source state.

On Completion – evaluates when the action completes. These transitions are ignored if no action is specified for the source state.

Note: On Completion transitions are taken only if the state is still active when the action completes. These transitions are ignored if the workflow has transitioned to another state.

Manual – evaluates when a user selects the transition in the UI and specifies parameters. The same action can occur using ectool or the Perl API by calling `transitionWorkflow`. Only users who have "execute" permission on the transition are allowed to use the manual transition.

Similar to other objects in the Commander system, workflow objects can contain access control privileges, properties, and parameters.

Workflow objects

Workflow objects are split into two types: **Definition** objects and **Instance** objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.

When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.

Note: We omit the "Instance" qualifier for brevity in the API and the UI.

Workflow Definition

This is the top-level workflow object, which is a container for states, and transitions, and other information defining your workflow. In the same way a procedure defines the behavior of a job, the workflow definition defines the behavior of a workflow.

Create a project to get started:

- Select the Projects tab, then click the **Create Project** link. After creating a project, its name will appear in the table on the Projects page.
- Select your project name to go to its Project Details page, select the Workflow Definitions subtab, then select the **Create Workflow Definition** link.
- On the New Workflow Definition page, name your workflow definition. Access the **Help** link on that page if you need help creating the Workflow Name Template (similar to the Job Name Template). Click **Save**.
- Now, on the Workflow Definition Details page (Graph view), you are ready to create the first state for your workflow.

Click [here](#) to see the help topic for the Workflow Definition Details page that contains an example of how this page might look as you add states and transitions to your workflow definition. Notice also that you can add properties to the workflow definition at this point.

State Definition

Each workflow can contain one or more states. Defining states for a workflow is analogous to defining steps for a procedure.

You might choose to name your states for their intended purpose (for example: waiting for user, building, testing, or test1, test2, and so on) just as you might name steps for your procedure. You can define multiple states as "startable" so you can choose different starting points when you begin using the workflow. By default, the first defined state is always "startable".

As you create states, they will be displayed in the graph on the Workflow Definition Details page.

Selecting a state in the Graph view opens the State Definition panel to define:

- the action you want this state to perform, along with any parameters required by the action,
- transitions for this state,
- parameters for this state,
- email notifiers for this state,
- and any properties you want to assign to this state.

Transition Definition

Each state can contain one or more transitions. The transition definition requires a name for the transition. This transition name will appear on the Workflow Definition Details graph and the Workflow Definition Details List view.

You can define one or more transitions for each state.

When defining a transition, you can:

- add a text description for your reference,
- select the "target" state,
- select the type of transition trigger you want to use (On Enter, On Start, On Completion, or Manual) See the [Workflow Definition Details](#) Help topic for trigger type definitions and other information for defining a transition,

- select (or supply) a Condition to specify when the transition is allowed to trigger,
- and assign parameters to a transition. These parameters are passed to the target state.

Workflow

While we generically refer to or think of a "workflow" as an automated process to perform multiple tasks, it is actually the end result of running a Workflow Definition. Running a Workflow Definition produces a Workflow, which is analogous to running a Procedure, which produces a Job.

After you start running Workflow Definitions, select the main Workflow tab to see all workflows that ran (or are still running) listed in a table, each linked to its own Workflow Details page. The Workflow Details page main view is a graphical representation of the workflow, and also provides links to stop the workflow process, run the workflow again, see the workflow log, and more.

A Workflow contains a pointer to:

- its workflow definition,
- sets of states and their transitions,
- the current and initial state,
- parameters to the initial state,
- a "complete" value to specify whether or not the workflow is complete,
- and a log containing transitions taken and user events.

State

The run-time instance of a State Definition is a State. The state is part of a workflow and contains information about actual parameter values specified by the user when the workflow was run or from transitions taken to enter the state.

If the state has an *action*, then the state contains information about the most recently created job or workflow. Because a state may be entered multiple times, the state contains a copy of the state definition properties needed to launch the action again.

Note: After the state is created, it no longer depends on the state definition, so any changes made to the definition will not affect workflows already running. Changes do not take effect until the next time the workflow runs.

Transition

The run-time instance of a Transition Definition is a Transition. The transition is part of a state and contains a copy of information from the transition definition. Each transition object corresponds to a transition definition, but once defined, each transition is recognized by its unique name. A transition contains unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry, the Javascript condition, and the trigger type instructing the transition when to occur.

Note: After the transition is created, it no longer depends on the transition definition, so any changes made to the definition will not affect workflows already running. Changes do not take effect until the next time the workflow runs.

A transition moves the workflow from its state to a *target* state.

Access Control

Setting access control privileges for those persons who can or cannot run a workflow or execute a manual transition may relate to security policies in your organization.

Specific workflow access control notes:

- All workflow, state, and transition ACLs are copied from the definition object to their corresponding instance when the workflow is run initially.
- To run a workflow, you must have execute permission on the state definition you are using to start the workflow.
- To take a manual transition, you need execute permission on that transition.
- To strictly control who can take a manual transition:
Grant read and modify privileges to anyone who can view or edit the workflow, but only give execute permission to those users or groups who can take the manual transition.
- You can control who can start each of the startable states by setting ACLs on state definitions accordingly.

For more information about access control, including examples you might use, see the [Access Control](#) Help topic.

Property Search Paths

A workflow serves as a hub of information that can be shared between states, transitions, jobs, and other workflows.

Depending on the context, an implicit search path is used to reference a property by name. Using a simple property name like `$(someCustomProperty)` “walks” down the following paths until it finds a property by that name. If the search fails to find a property, an error is produced.

The search path followed to resolve a simple property name depends on where the reference occurs:

Transition context:

1. Transition property sheet
2. State property sheet – includes parameter values passed to that state and any custom properties. Because a state can be entered multiple times, parameter values are available for the most recent entry only.
3. Starting state property sheet – used to get to parameter values for the initial entry into the workflow.
4. Workflow property sheet – used to access information shared across the workflow.

State context:

1. State property sheet – includes parameter values passed to that state and any custom properties. Because a state can be entered multiple times, parameter values are available only for the most recent entry.
2. Starting state property sheet – used to get to parameter values for the initial entry into the workflow.
3. Workflow property sheet – used to access information shared across the workflow.

For information on context-relative property path shortcuts that allow passing information from one object to another or accessing a property from different parts of the workflow, review the [Properties](#) help topic. The Properties Help topic also includes all intrinsic workflow-related properties.

Parameters

States may use formal parameters to determine what information to provide when the workflow enters that state. Parameter values can come from the initial run workflow call or from a subsequent transition into the state.

An automatic transition must specify all values required by the target state. A manual transition may omit some values to be passed to its target state. If a manual transition does not specify all values needed by the target state, then any missing values are collected from the user when the transition is taken.

Parameters to the starting state for the workflow are collected when the workflow is launched. These parameters are accessible throughout the workflow and act like global workflow parameters. See the [Property Search Paths](#) section above.

It is important to distinguish between formal parameters on a state and formal parameters on the action the state calls (procedure or workflow starting state). The procedure (or workflow) may have its own parameters, so the state is responsible for supplying required actual parameter values. In some cases, values may be static values, in other cases the values may come from the state's parameters or those of the starting state (for example, global workflow parameters)

Parameters on the calling state do not need to be the same as the parameters on the action:

- A state parameter value can be passed as the value for an action's parameter by specifying a property reference in the value field on the Action tab for the state definition.
- All parameters on workflow current and starting states are accessible with a simple property reference (for example, '\$[param1]')
- Parameters on a workflow starting state act like global workflow parameters because they are available everywhere within the workflow.

Sending Notifications

A state can be defined to send email notifications when it is entered, when it starts its action, and/or when the action completes. Notifiers are created by visiting the Notifications tab on the State Definition panel. See the [Email Notifier](#) Help topic for details on defining email notifiers. Sample notifier templates are provided also.

Workflow Logs

Each workflow keeps a log of events that relate to the workflow. The log contains information about the evaluation of transition conditions, transitions taken, actions started, and other information that might prove useful when debugging a workflow. Access the log by clicking the **View Log** link on the Workflow Details page. **For more information**, click [here](#) to go to the Workflow Log help topic, which contains a screen example of the Workflow Log page.

Visualizing a Workflow

Two Workflow web pages, Workflow Definition Details and Workflow Details, open to the Graph view.

- Workflow Definition Details page - As you build your workflow, you can see a visual representation of state and transition definitions you define.
- Workflow Details page - displays a graph, updated in real-time, for your running or completed workflow.

Your workflow Graph (on either page) contains context-sensitive links to various options. Options are different depending on whether or not you are building or running your workflow. Workflow Definition Details and Workflow Details help topics include information about the context-sensitive link options provided within the workflow graph.

The next section, "Building a Workflow - a Tutorial," begins with a "sketched" representation of the workflow the tutorial will create. At the end of the tutorial, see the actual workflow graph created by the Commander workflow process.

Building a Workflow - a Tutorial

This over-view tutorial is an example of how to create a "build-test-approve" workflow, building each state and transition we want the workflow to include, and building the workflow in the order we need to use to achieve our purpose. The diagram below outlines our workflow process.

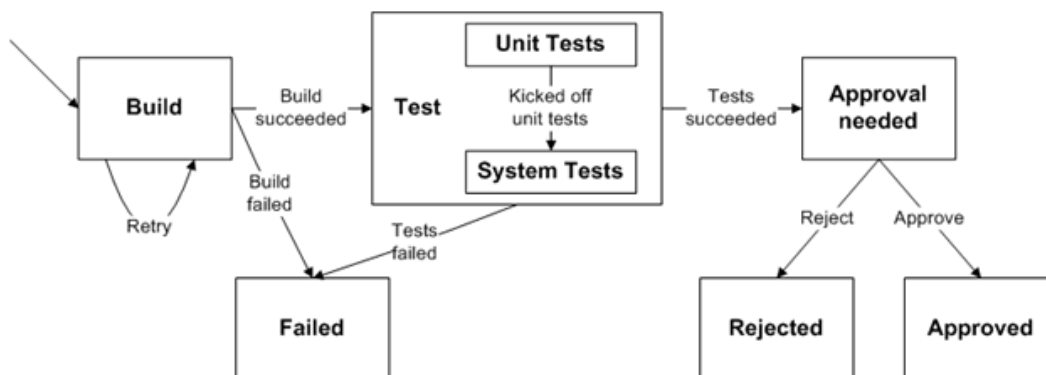
You can read through the tutorial or actually recreate this example workflow on your machine. If you want to use this tutorial as a real, working example, you must do some initial setup first:

- Create a project named "Sample"
- Create a procedure name "BuildProduct"
- On the "BuildProduct" procedure, add a parameter named "branch"

The following diagram is a visual illustration of the workflow we are going to build.

Reading the diagram:

- the starting state is *Build*, which could transition to the *Test* state or the *Failed* state
- two test states - *Unit Tests* and *System Tests*, which could transition to a *Failed* state or an *Approval needed* state
- the *Approval needed* state could transition to one of two states - *Rejected* or *Approved*
- all transitions are represented by the lines between the states, which include "arrows" to show the possible workflow directions (depending on the outcome of each state's action)



Best Practice tip

Designing your workflow on paper or whiteboard, creating your own state and transition diagram, is a great way to organize your thoughts and map your workflow process.

A summary of the component sections to build our workflow:

- [Calling a job from within a workflow](#)
- [Collecting a parameter when a workflow is launched and passing its value to a job](#)
- [Retrying a state](#)
- [Automatically transitioning to different states based on the outcome of a job](#)
- [Invoking another subworkflow](#)
- [Running jobs in parallel](#)
- [Automatically transitioning to different states based on the outcome of jobs in a workflow](#)
- [Waiting for manual intervention](#)
- [Restricting who can take a manual transition](#)
- [Sending email notifiers](#)
- [Adding a global parameter to use later in the workflow](#)
- [Setting the name of your workflow](#)
- [Workflow list](#)

To begin...Calling a job from within a workflow

Or, running a job by calling its procedure from a workflow state.

- Select the Projects tab, select the Sample project to go to the Project Details page, then select the Workflow Definitions subtab.
- Click the **Create Workflow Definition** link to go to the New Workflow Definition page..
- Create a new Workflow Definition named *Lifecycle*.



Project: Sample

New Workflow Definition

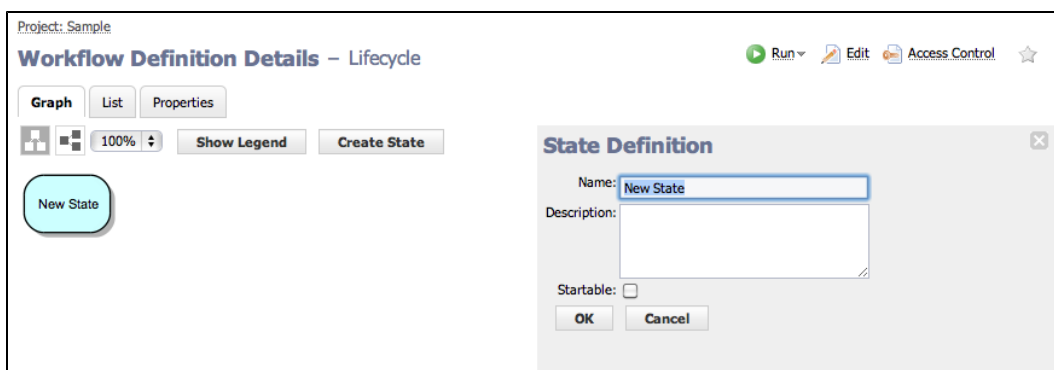
Name:

Description:

Workflow Name Template:

OK Cancel

Because the workflow definition, *Lifecycle*, is a new definition and does not have any state or transition definitions, the State Definition panel opens, ready for you to create a new state definition for your workflow.



Project: Sample

Workflow Definition Details – Lifecycle

Run Edit Access Control

Graph List Properties

100% Show Legend Create State

New State

State Definition

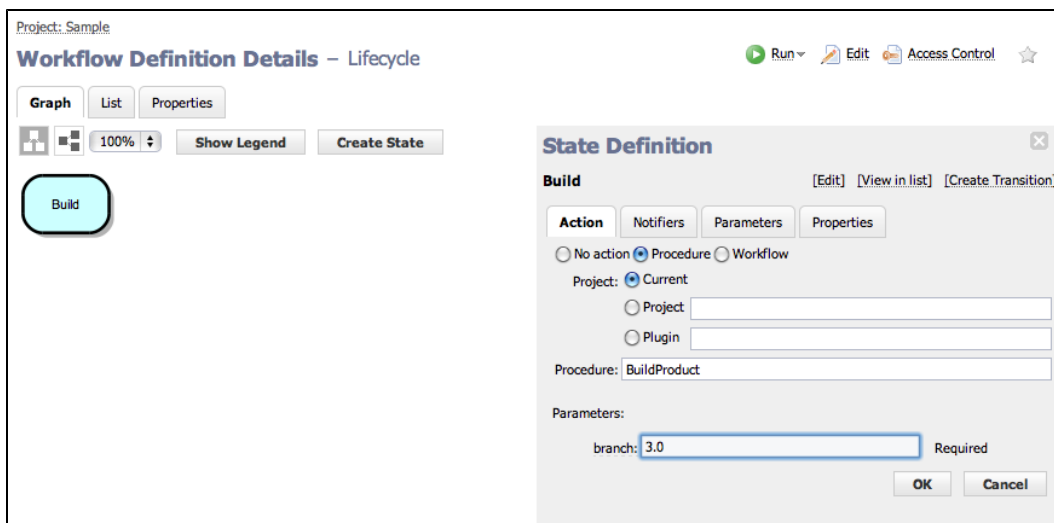
Name:

Description:

Startable: ☐

OK Cancel

- Change the system-generated state Name to *Build* and click **OK**.
- The State Definition panel is displayed.



Project: Sample

Workflow Definition Details – Lifecycle

Run Edit Access Control

Graph List Properties

100% Show Legend Create State

Build

State Definition

Build [Edit] [View in list] [Create Transition]

Action Notifiers Parameters Properties

☐ No action ☒ Procedure ☐ Workflow

Project: ☒ Current ☐ Project

☐ Plugin

Procedure:

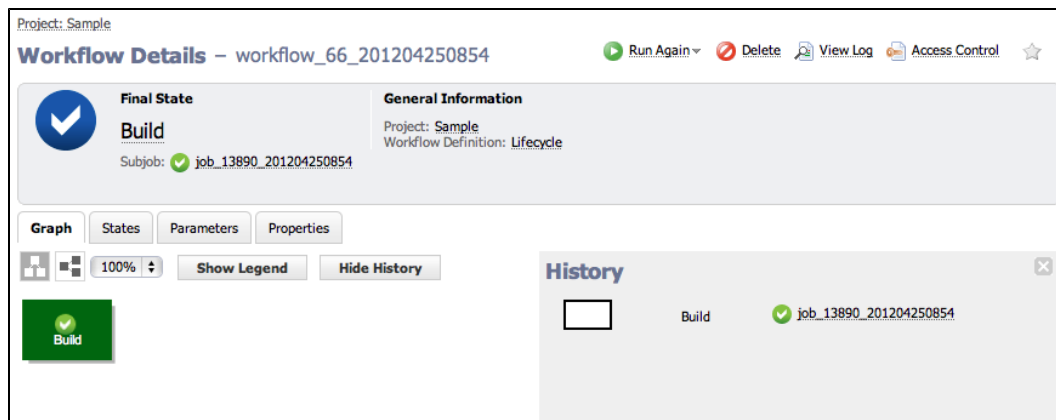
Parameters:

branch: Required

OK Cancel

- Specify the Action for this state, which is Procedure.
In this example, the procedure *BuildProduct* already exists with a single parameter named *branch*.

- Set the value of this parameter to "3.0".
- The *Build* state now has an Action associated with it.
- Click **OK**.
Note the shape of this state in the graph. The state shape will be different depending on the state Action selected.
- Click **Run** to launch the workflow.

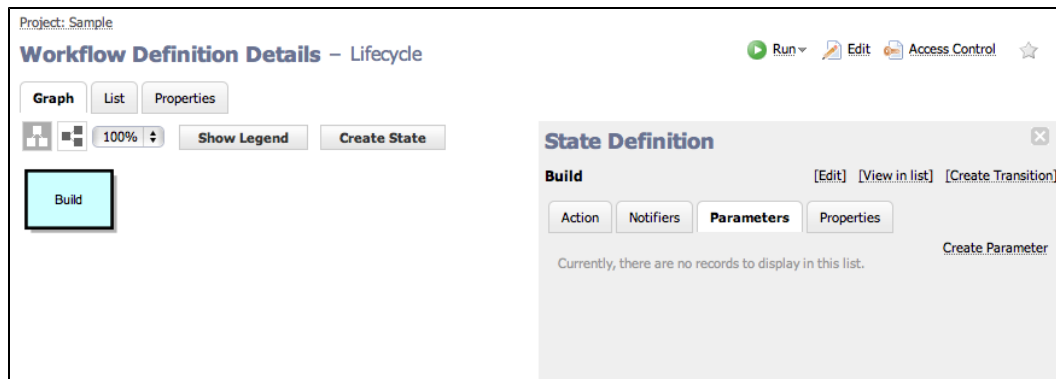


- Running the workflow shows a Job was launched by the *Build* state. The workflow job completed successfully, indicated by the "checkmark" on the state.
- You can get more details about the job by clicking on the *Build* state, which opens the State Details panel, and clicking on the Job link displays the Job Details page.
- Alternately, clicking on the Show History button opens the History panel to show what ran and the workflow outcome.

Collecting a parameter when a workflow is launched and passing its value to a job

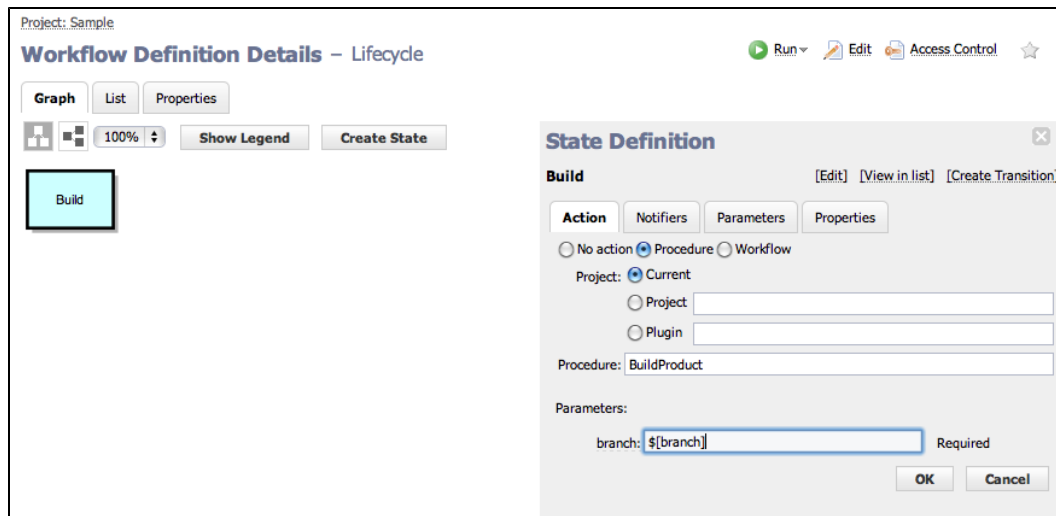
Or, "prompting" for a parameter value when a workflow is launched and passing its value to a procedure.

- Return to the Workflow Definition page for the **Lifecycle** workflow.
- Click the *Build* state (from the graph) to open the State Definition panel, then select the Parameters tab.



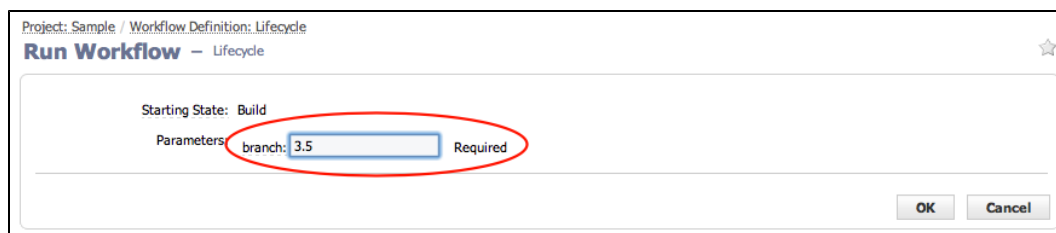
- Click the **Create Parameter** link to display the New Parameter page.
- Create a parameter called "branch".
- Add a description and make sure the Required? box is "checked" and click **OK**.

- On the Workflow Definition Details page again...
- Change the value passed to the job from 3.0 to `${branch}`.



Now, when the workflow is launched, you will be prompted for the value of *branch* and you can specify a different value if necessary.

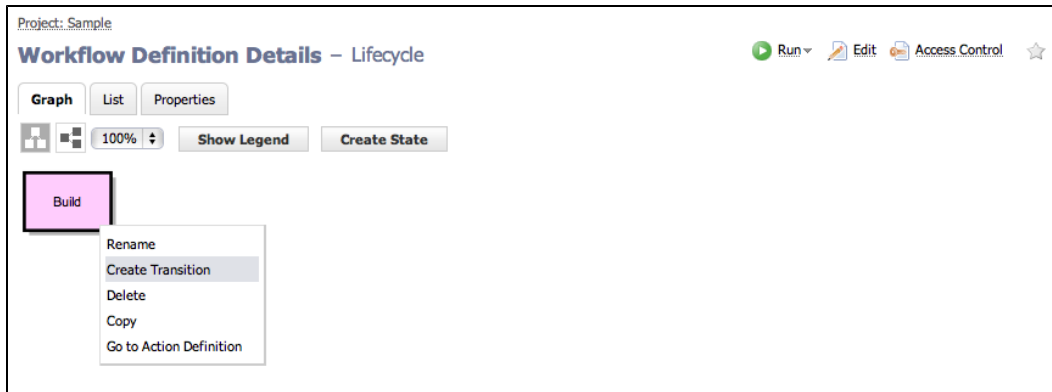
For example, a "3.5" value is now specified in the screen below.
The value is passed to the job when the workflow is launched.



Retrying a state

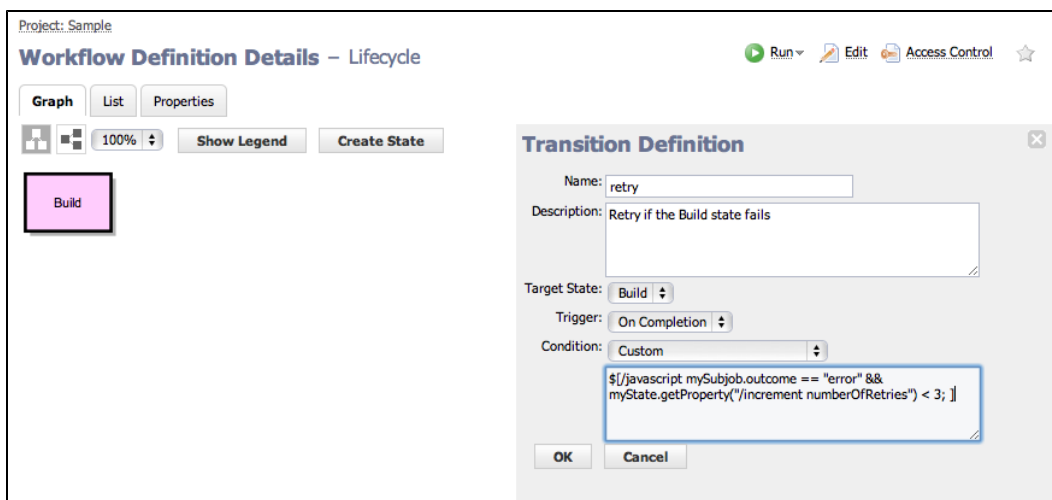
If a State fails (technically, the job started by the workflow state), you may choose to retry it a few times before giving up.

- Create a new transition from the *Build* state back to itself.
- Right-click on the *Build* state and select the **Create Transition** link.

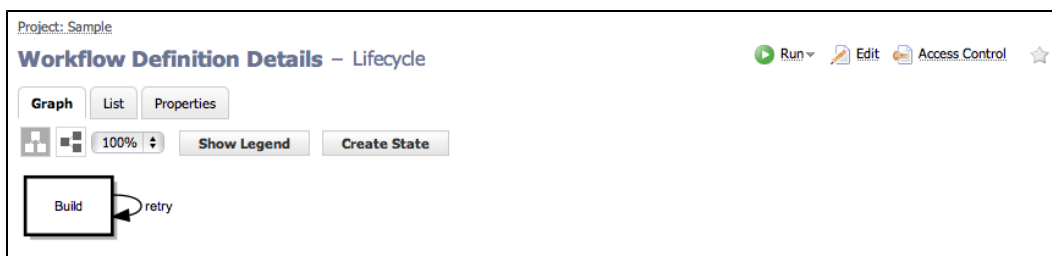


- Create a new On Completion transition named "retry" from the *Build* state that returns to the same state when the job fails.
- Add a limit of 3 retries by using the following JavaScript condition:

```
$/javascript mySubjob.outcome == "error" && myState.getProperty('/increment numberOfRetries') < 3; ]
```



- The new transition is displayed on the Workflow Definition Details page.



- When we run the workflow, we see only the last job that was run.

Project: Sample

Workflow Details – workflow_82_201204251214

Run Again Delete View Log Access Control

Final State

Build

Subjob: job_14211_201204251214

General Information

Project: Sample

Workflow Definition: Lifecycle

Graph States Parameters Properties

100% Show Legend Show History

Build retry

- Clicking the States tab and then the **Show All** link displays all jobs that were invoked by the *Build* state.

Project: Sample

Workflow Details – workflow_82_201204251214

Run Again Delete View Log Access Control

Final State

Build

Subjob: job_14211_201204251214

General Information

Project: Sample

Workflow Definition: Lifecycle

Graph **States** Parameters Properties

State Name	Action
Build	job_14211_201204251214 [Show All]

On this screen, you can see the build was attempted 3 times because that was the limit we specified in the JavaScript condition for the "retry" transition.

Job Search Results

3 Results for "callingStateId" equals "282"

New Search Save Filter Edit Search Refresh Search

Job	Status	Priority	Procedure	Launched By	Elapsed Time	Start Time	Actions
job_14211_201204251214	Error	normal	Sample:BuildProduct	workflow_82_201204251214: Build	00:00:00.453	2012-04-25 12:14:18 PDT	Delete
job_14210_201204251214	Error	normal	Sample:BuildProduct	workflow_82_201204251214: Build	00:00:00.613	2012-04-25 12:14:17 PDT	Delete
job_14209_201204251214	Error	normal	Sample:BuildProduct	workflow_82_201204251214: Build	00:00:00.539	2012-04-25 12:14:17 PDT	Delete

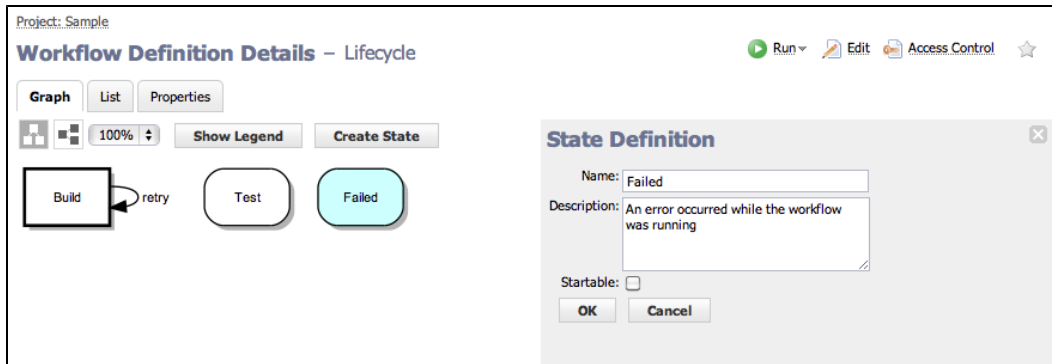
Records per page: 20 1 thru 3 of 3

Also, you can see the number of times a state was visited and which transitions were taken by clicking the **Show History** button.

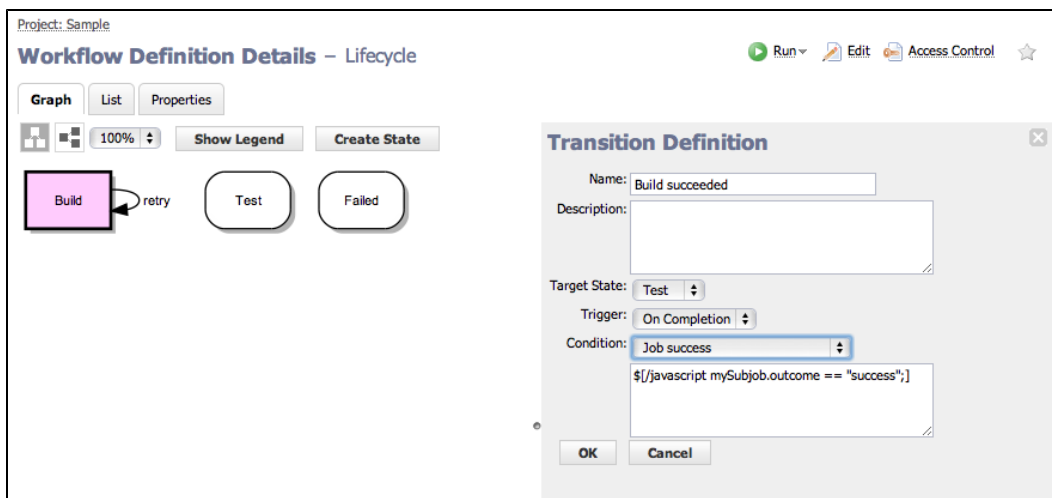
Automatically transitioning to different states based on the outcome of a job

If the *Build* job fails, we might want to end the workflow. If the job succeeds, we might want to continue to the next stages of the workflow.

- Add new states to transition to, depending on the outcome of the job:
 - Create the *Test* state where the workflow will transition to when the build state succeeds.
 - Create a *Failed* state.

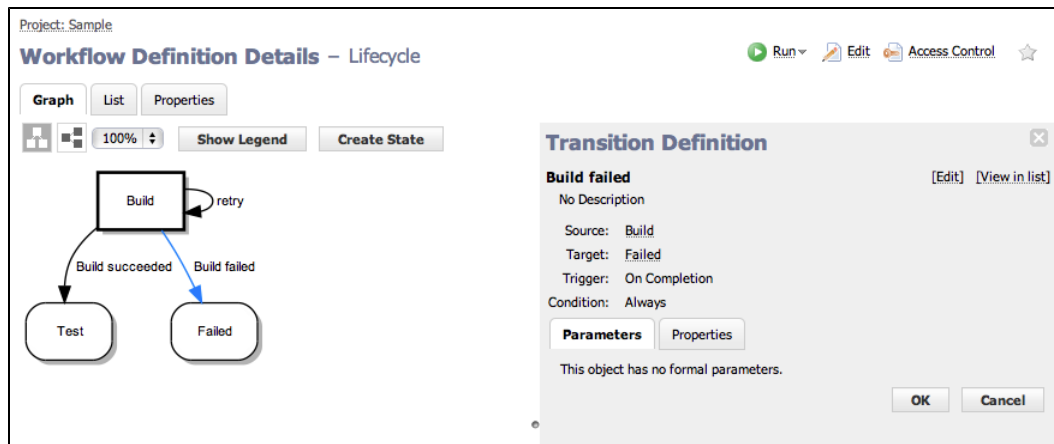


- Create an On Completion transition from *Build* to *Test* when the job succeeds.
- Notice that the Condition drop-down menu provides a template condition called *Job success* that can be used here.

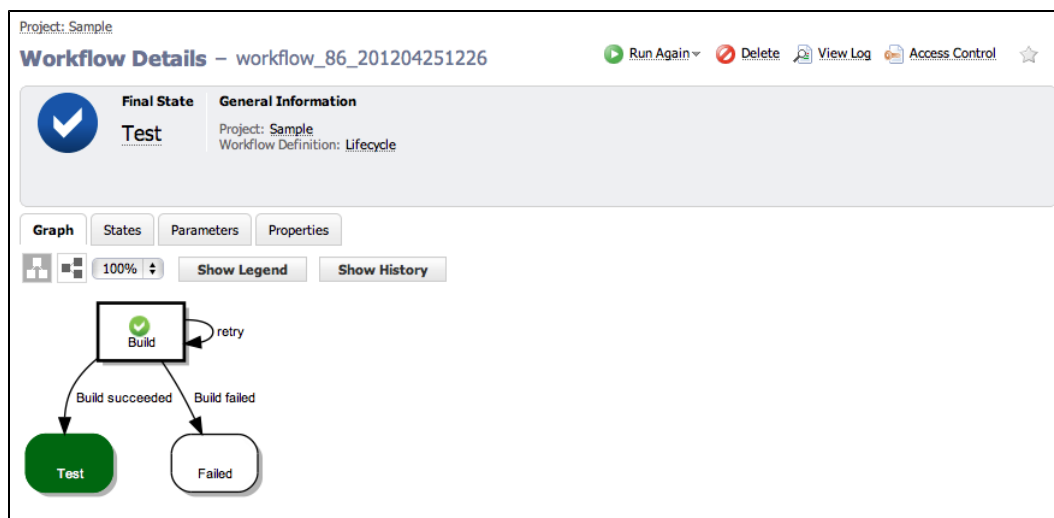


- Create an On Completion transition from *Build* to *Failed* when the job fails.
- Notice that the Condition field remains blank.
Because transitions are evaluated in order, if the workflow gets to this transition, we will know that the "success" case will be false.

Leaving the Condition field "Always" allows us to handle all "non-successful" outcomes (error or warning).



- When you run the workflow, if the build succeeds the workflow will end up in the *Test* state.



- The workflow will end up in the *Failed* state if the build fails after trying three times. (In this example, we show the Workflow History panel open.)

Project: Sample

Workflow Details – workflow_87

Run Again Delete View Log Access Control

Final State **Failed**

General Information
Project: Sample
Workflow Definition: Lifecycle

Graph States Parameters Properties

100% Show Legend Hide History

```

graph TD
    Build[Build] -- retry --> Build
    Build -- "Build succeeded" --> Test[Test]
    Build -- "Build failed" --> Failed[Failed]
  
```

History

Build	job_14240_201204251227
retry	
Build	job_14241_201204251227
retry	
Build	job_14242_201204251228
Build failed	
Failed	

Invoking another workflow

Just as a state can invoke a job as its “Action”, it can invoke another workflow, which then becomes its subworkflow.

- From the Workflow Definition Details page for *Lifecycle*, click on the *Test* state in the graph.
- Specify the “Action” for the Test state, which is a Workflow.
- In this example, the workflow *AutomaticTests* already exists and so does a state called *UnitTests*, which has a single required parameter called *buildName*.
- We can get the name of the build from the *Build* state by using the property path `$/myWorkflow/states/Build/subjob/jobName`.

Project: Sample

Workflow Definition Details – Lifecycle

Run Edit Access Control

Graph List Properties

100% Show Legend Create State

```

graph TD
    Build[Build] -- retry --> Build
    Build -- "Build succeeded" --> Test[Test]
    Build -- "Build failed" --> Failed[Failed]
  
```

State Definition

Test [Edit] [View in list] [Create Transition]

Action Parameters Notifiers Properties

☐ No action ☐ Procedure ☒ Workflow

Project: ☒ Current

☐ Project

☐ Plugin

Workflow:

Starting State:

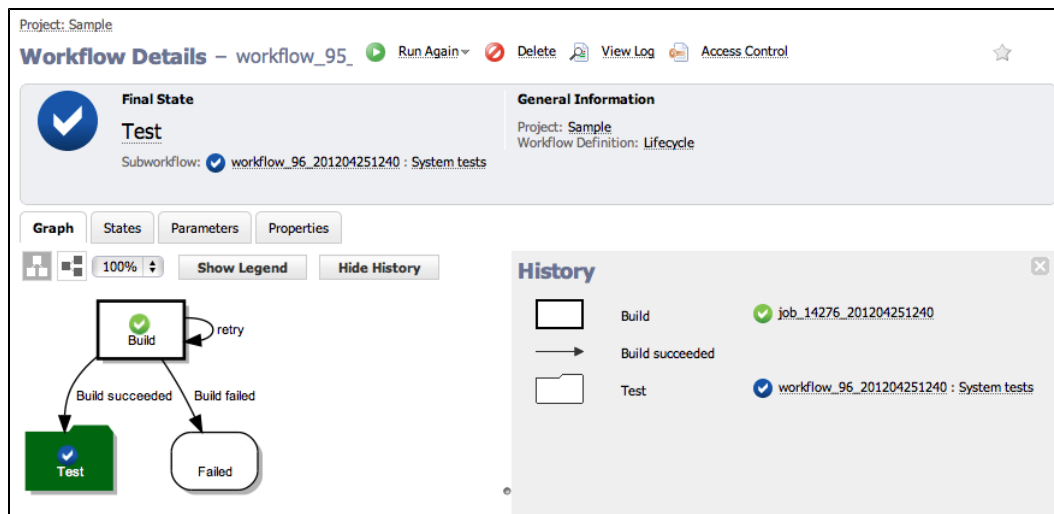
Parameters:

buildName:

Saved

OK Cancel

- The *Test* state now has an Action associated with it.
- Because the *Test* state Action is Workflow, the object shape in the graph has changed.

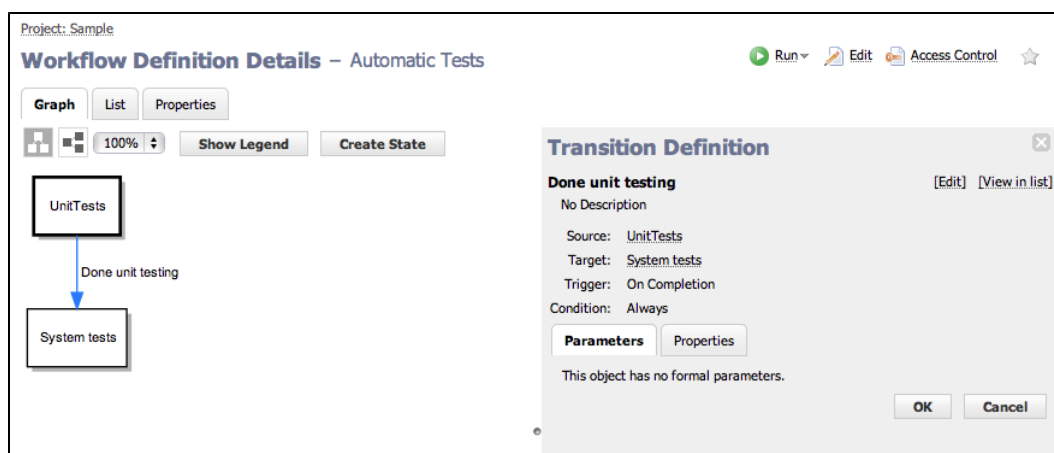


- Running the workflow shows another workflow was launched by the *Test* state.
- Click on the workflow link in the History panel to see more details about that workflow.

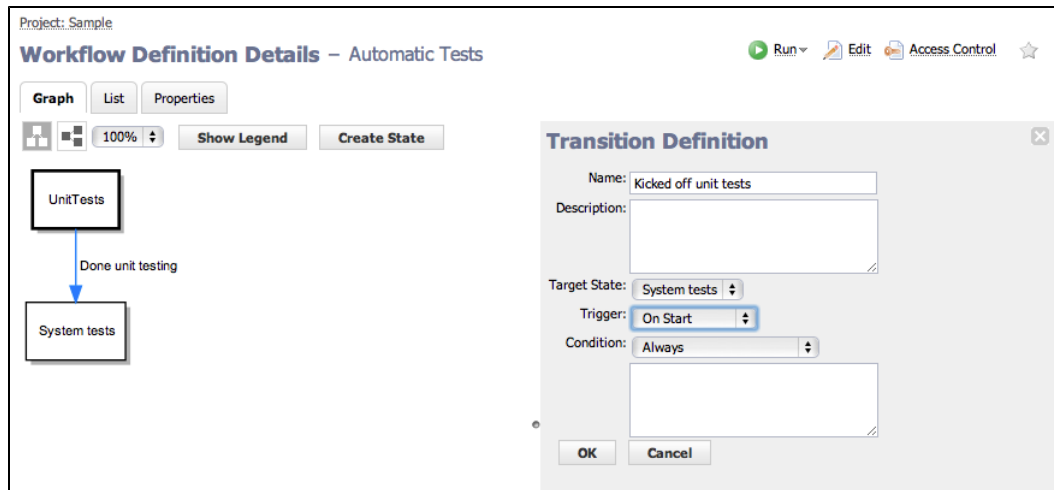
Running jobs in parallel

Run multiple jobs at the same time.

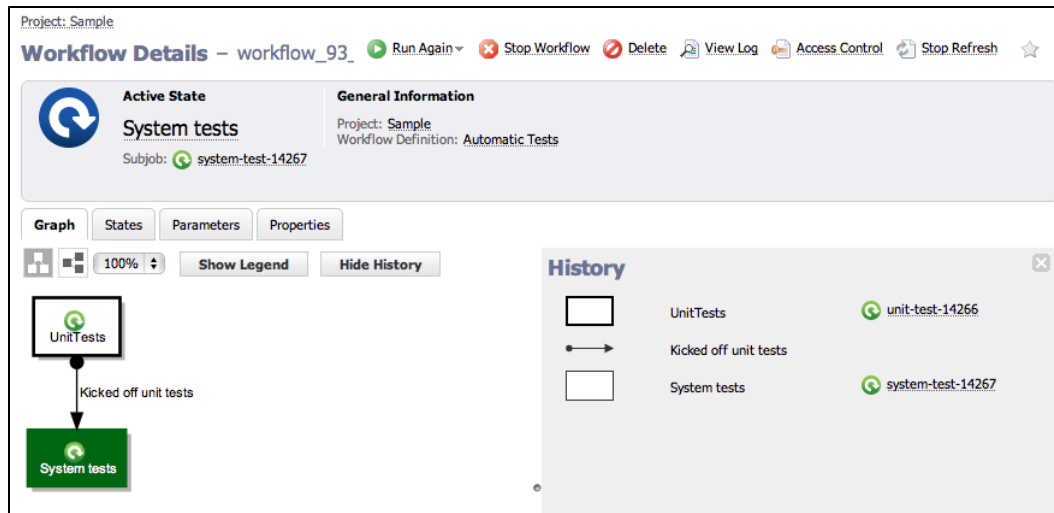
- Look at the *AutomaticTests* workflow that was invoked (in the previous example) by the *Test* state in our main *Lifecycle* workflow. The first state in this workflow runs unit tests, and when the tests complete, the state automatically transitions to the second state that runs system tests.



- Now we will change the behavior of the transition to move from the *Unit tests* state to the *System tests* state as soon as the first job is launched.
- Edit the existing transition by renaming it to *Kicked off unit tests* and changing it to an On Start trigger type.



- Now, when the *AutomaticTests* workflow is launched, both jobs will run in parallel.
- The workflow completes automatically when both jobs complete.



Automatically transitioning to different states based on the outcome of jobs in another workflow

If the jobs in the *Test* workflow fail, we will want the workflow to end. If the jobs succeed, we will want to continue to the next stage(s) of a calling workflow.

- Click the **Create State** link.
- Create the *Approval needed* state that the workflow will transition to when tests succeed.
- Create a transition named "Tests succeeded" from *Test* to *Approval needed* when all jobs in the workflow succeed.
(Right-click on the *Test* state to create the transition.)
- The following JavaScript condition checks the outcome of all subjobs for a subworkflow and evaluates to "true" only if they all succeeded:

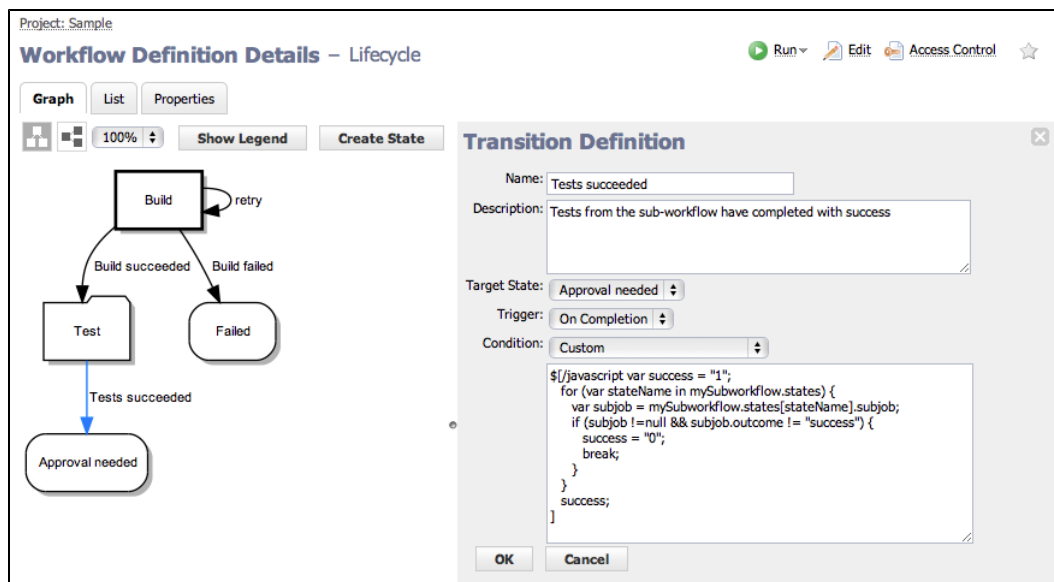
```

$[/javascript
  var success = "1";

  for (var stateName in mySubworkflow.states) {
    var subjob = mySubworkflow.states[stateName].subjob;
    if (subjob !=null && subjob.outcome != "success") {
      success = "0";
      break;
    }
  }

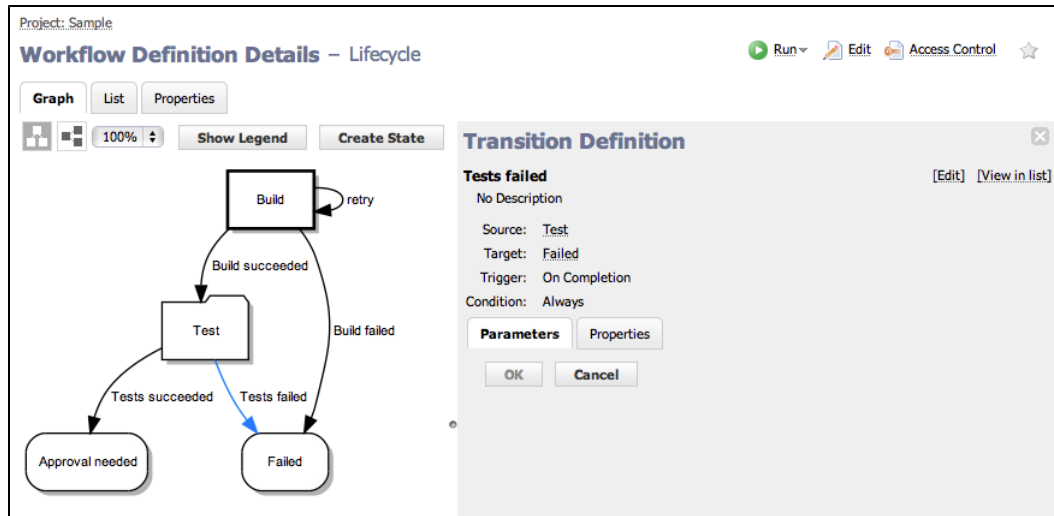
  success;
]

```

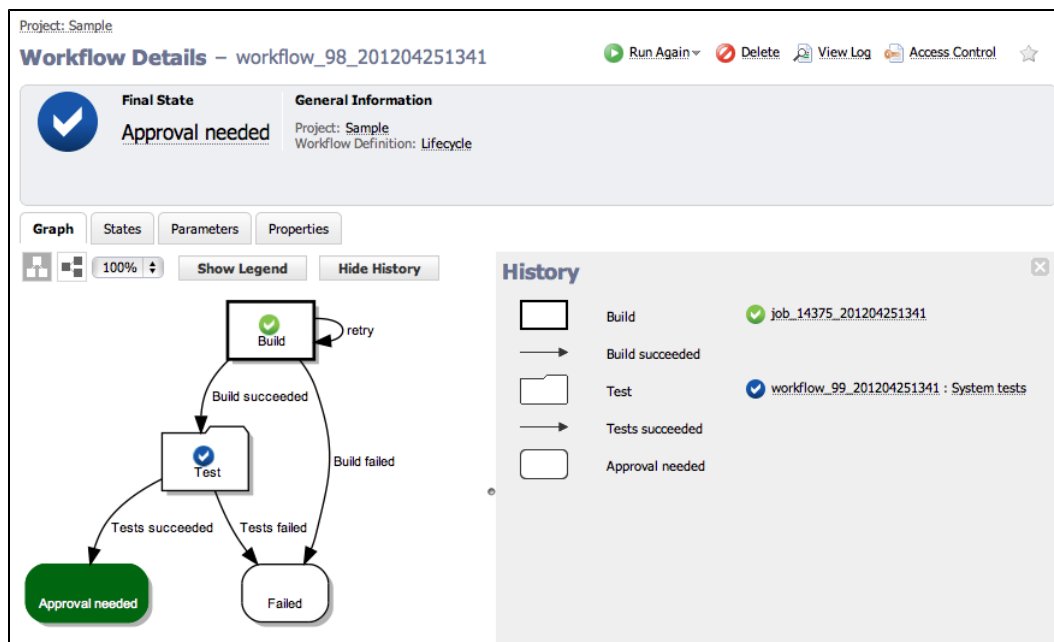


Note: We created the *Approval needed* state first, then we created the *Tests succeeded* transition to connect the two states.

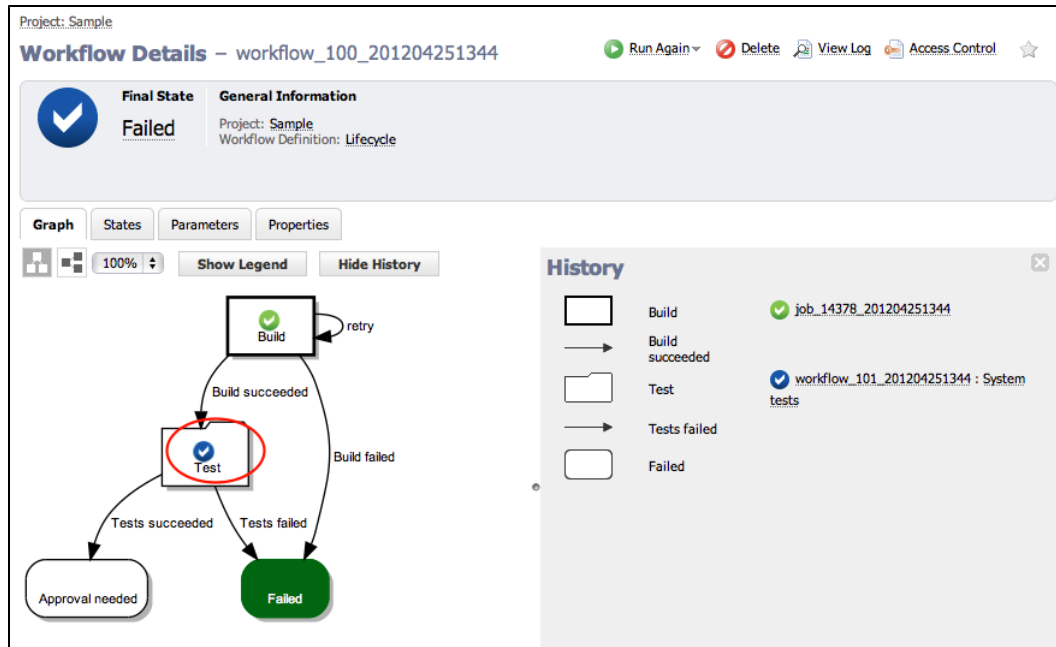
- Create a transition from the *Test* to the *Failed* state when either of the jobs in the workflow fails.
- Notice the Condition field remains blank.
- Because transitions are evaluated in order, if the workflow gets to this transition we know the success case was "false".



- When you run the workflow, if the test jobs succeed, the workflow will end up in the *Approval needed* state.

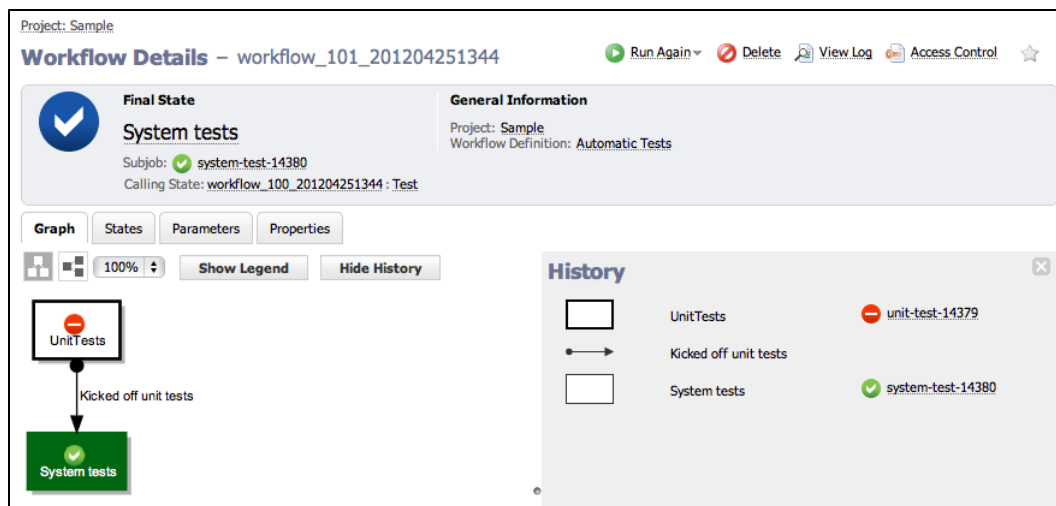


- In the following example, one of the tests in the *Test* workflow encountered an error, so the workflow will end up in the *Failed* state if either of the test jobs fail.



Note: The "blue checkmark" indicates the workflow (or subworkflow) has completed, which means it is no longer running. This icon does NOT indicate the workflow outcome (completed successfully, or with warnings or errors).

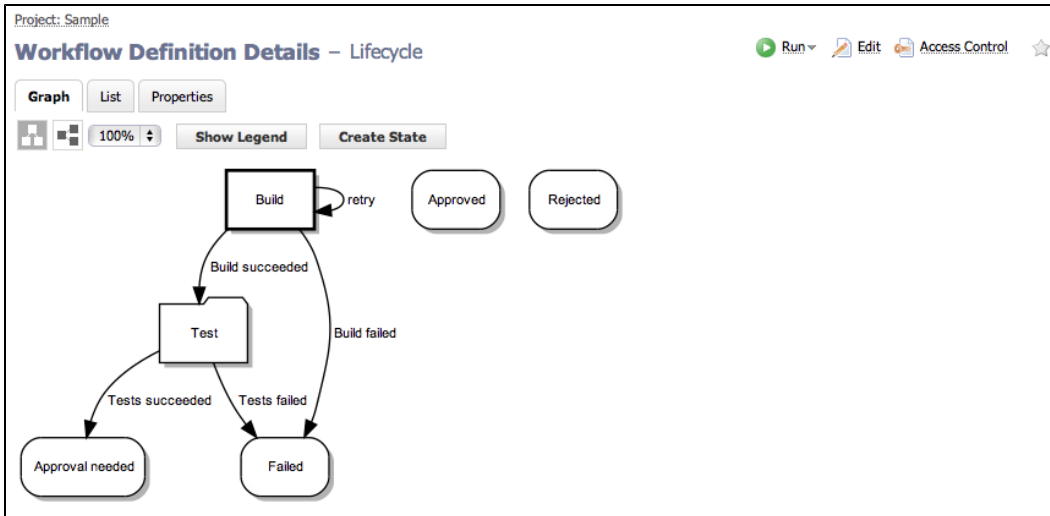
To see exactly what happened in the workflow, you can "drill-down" by clicking on the workflow link shown in the History panel.



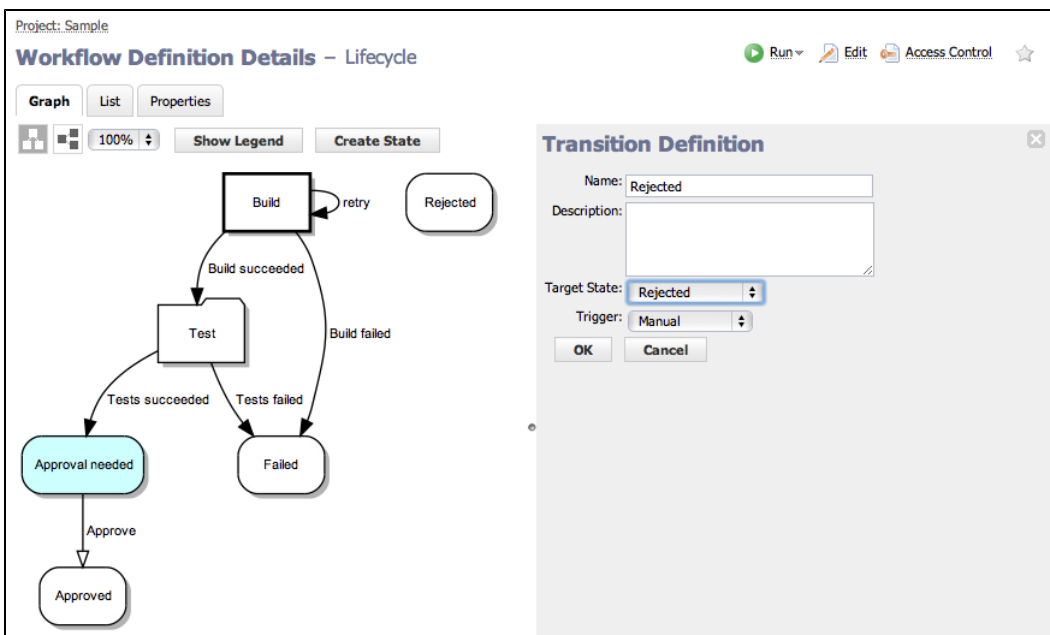
Waiting for manual intervention

Allow users to choose how to progress the workflow.

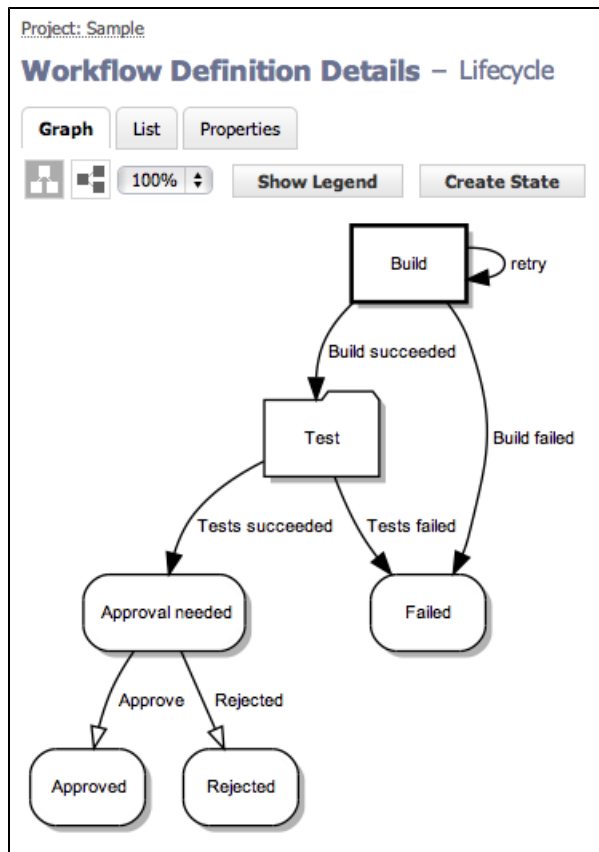
- From the Workflow Definition Details page for *Lifecycle*:
 - Create an *Approved* state and a *Rejected* state for the workflow to transition to when a user approves or rejects the build.



- For the *Approval needed* state, we create a manual transition called *Approve* that the user can select to progress the workflow to the *Approved* state.

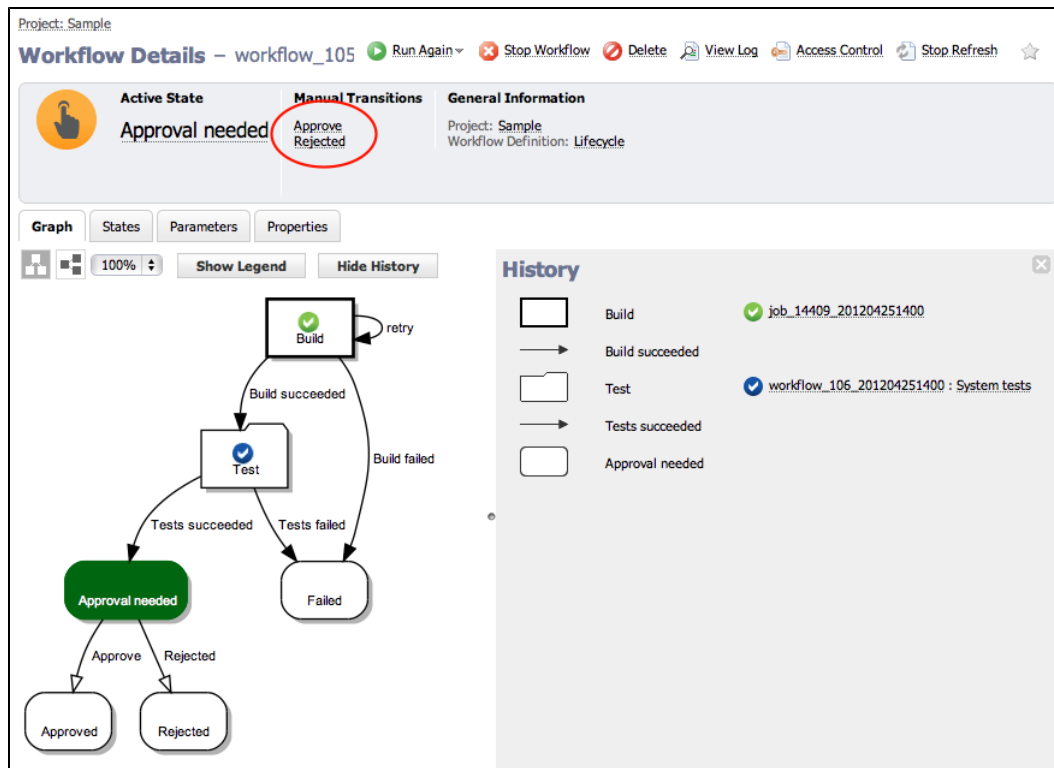


- Next, we create a manual transition called *Reject* that the user can select to progress the workflow to the *Rejected* state.
- At this point, our workflow now looks like the following example.

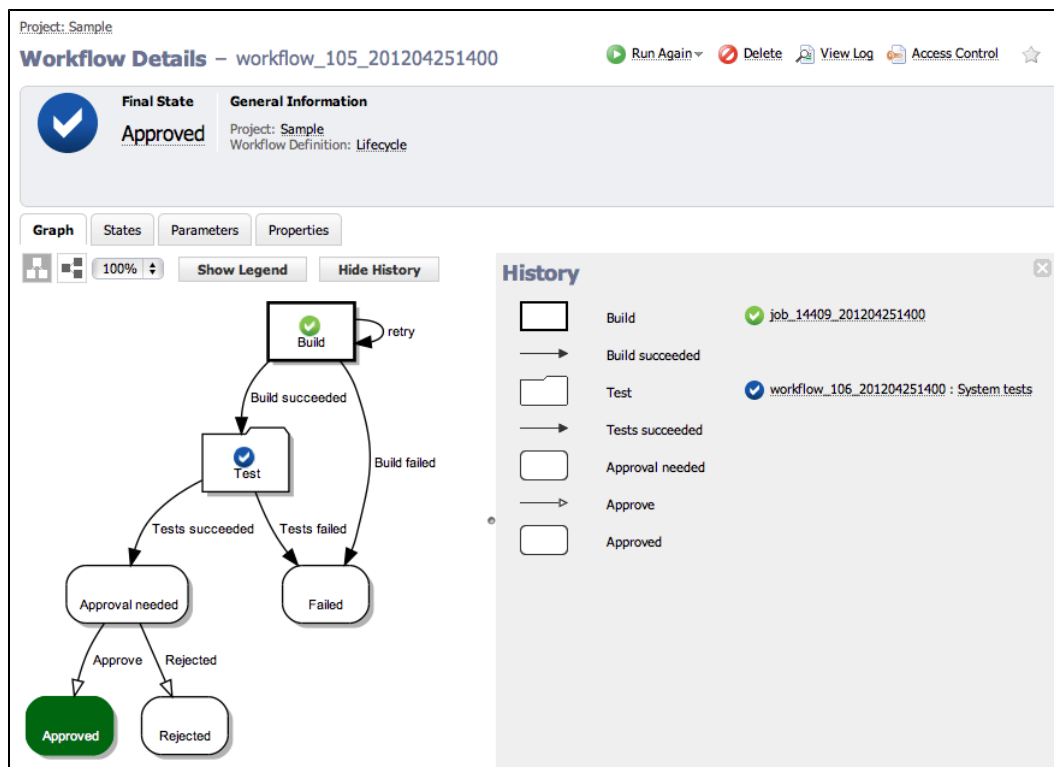


- When you run the workflow and the build and test phases succeed, the new manual transitions are made available to users on the Workflow Details page.

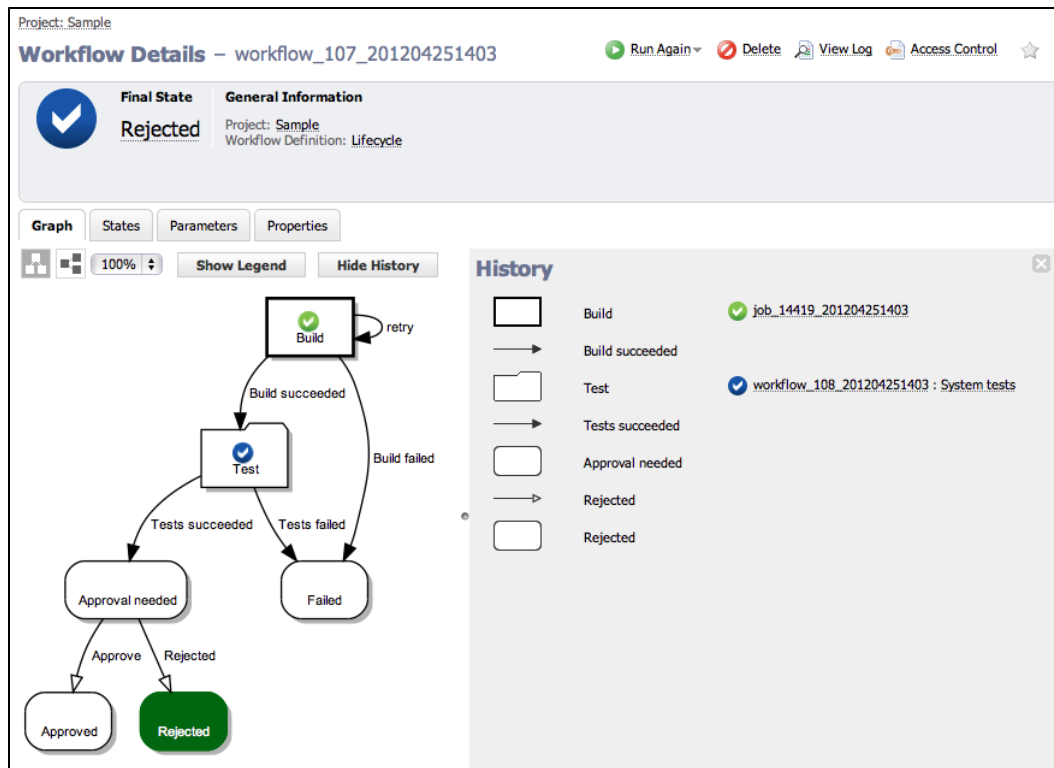
Note: If a user does not have execute permission for manual transitions, the Manual Transitions section will not be included in the summary section at the top of the Workflow Details page.



- If you choose the *Approve* transition, the workflow will end up in the *Approved* state.



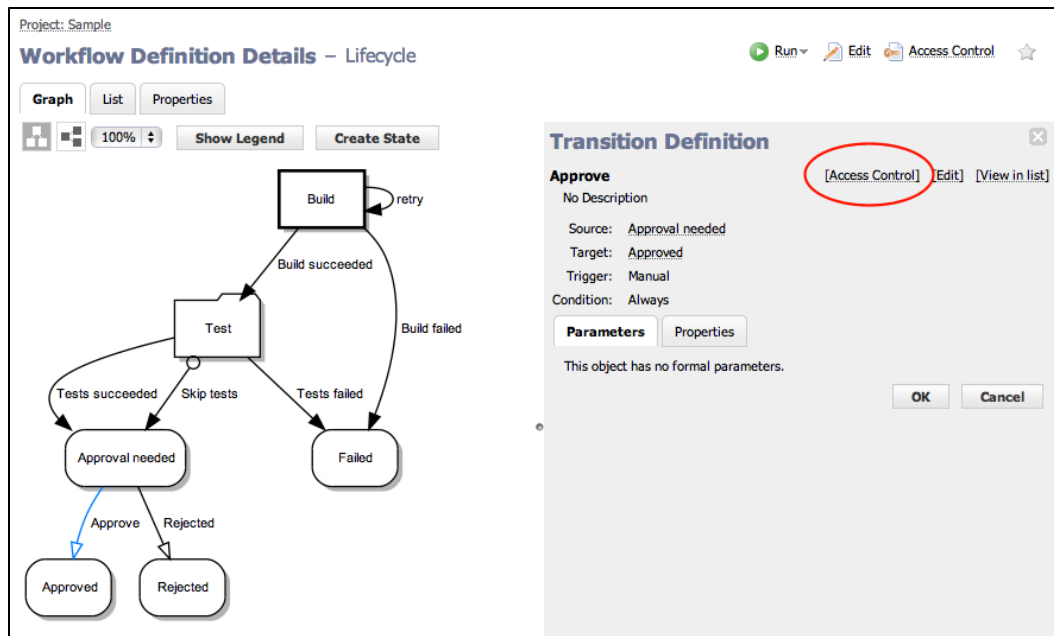
- If you choose the *Reject* transition, the workflow will end up in the *Rejected* state.



Restricting who can take a manual transition

Only allow certain users to take manual transitions.

- From the Workflow Definition Details page for *Lifecycle*:
 - When the workflow is created, access control entries from the Transition Definition are copied to the corresponding transition.
These are the access control entries for the *Approve* transition before we edit them (see below).
 - To add access control to a transition, select the transition or transition name.
 - When the Transition Definition panel opens, click the **Access Control** link,



- When the Access Control page is displayed, click the **Add Group** link to add privileges for a group that can take this transition.

Project: Sample / Workflow Definition: Lifecycle / State Definition: Approval needed / Transition Definition: Approve

Access Control – Transition Definition: Approve

Privileges for Transition Definition: Approve

Currently, there are no records to display in this list.

Add User | **Add Group** | Add Project | Break Inheritance

Privileges inherited from State Definition: Approval needed

Currently, there are no records to display in this list.

Privileges inherited from Workflow Definition: Lifecycle

Currently, there are no records to display in this list.

Privileges inherited from Project: Sample

Type	Name	Location	Read	Modify	Execute	Change Permissions
project	Sample		allow	allow	allow	allow

Privileges inherited from Projects

Privileges inherited from Server

Type	Name	Location	Read	Modify	Execute	Change Permissions
group	Everyone		allow	inherit	allow	inherit

Note: If you already have groups created, you can use them here or create two groups now: *Lifecycle Approvers* and *Lifecycle Modifiers*.

- Allow read and execute access for a group named *Lifecycle Approvers*. These users can see and take a manual transition in a workflow.
- Allow read, modify, and change permission access for a group named *Lifecycle Modifiers*. These users can edit the Transition Definition, but will not have access to take a manual transition in a workflow.

- "Break the inheritance" so only specified groups have access to this transition.
- Duplicate this access control configuration for the *Reject* transition.

Project: Sample / Workflow Definition: Lifecycle / State Definition: Approval needed / Transition Definition: Approve

Access Control – Transition Definition: Approve

Privileges for Transition Definition: Approve

Type	Name	Location	Read	Modify	Execute	Change Permissions	Actions
group	Lifecycle Approvers		allow	inherit	allow	inherit	Edit Delete
group	Lifecycle Modifiers		allow	allow	inherit	allow	Edit Delete

Transition Definition "Approve" disallows further inheritance.

- When a workflow is launched and reaches the *Approval needed* state, a user who is not a member of the *Lifecycle Approvers* group will not be allowed to execute any manual transitions.
- A user who is a member of the Lifecycle Approvers group can execute manual transitions.
- The Workflow Details page updates in real-time, which means you will see changes at the top of the page pertaining to the current state in the running workflow, and current changes in the graph also. For example, if the previous state had a manual transition, but the current state now processing does not have a manual transition, the Manual Transition section will not be displayed.

electriccommander

Logged in as 'Manager' Logout Help

Home Projects Jobs Workflows Cloud Artifacts Search Administration

Project: Sample

Workflow Details – lifecycle-3.5-workflow-8

Run Again Delete View Log Access Control

Final State Approved

General Information
Project: Sample
Workflow Definition: Lifecycle

Graph States Parameters Properties

100% Show Legend Hide History

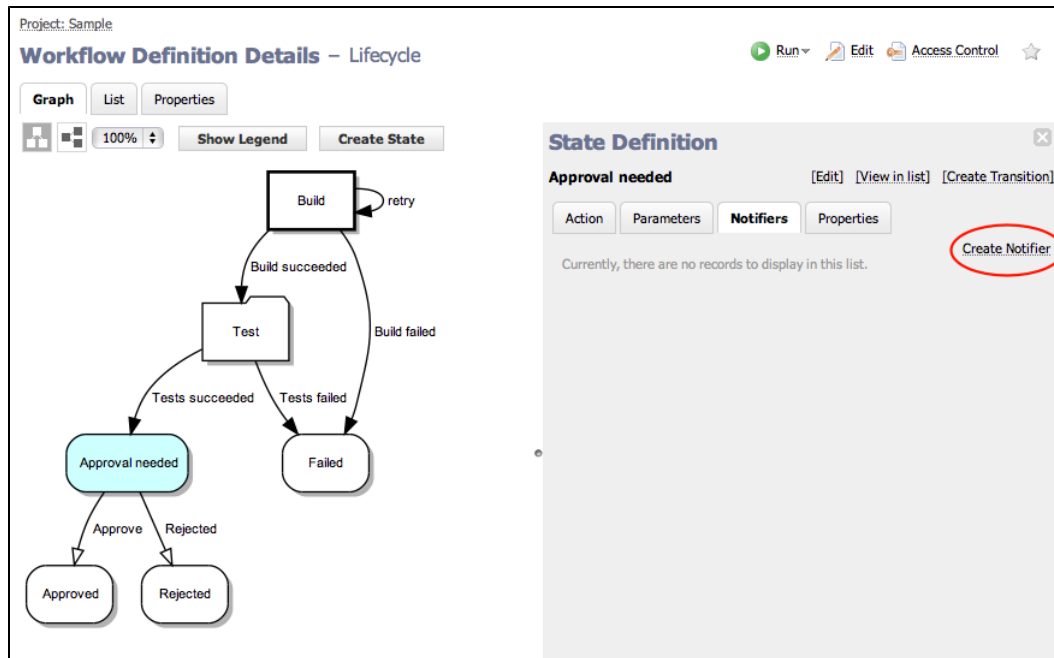
History

- Build job_19937_201204292333
- Build succeeded
- Test workflow_146_201204292333 : System tests
- Tests succeeded
- Approval needed
- Approve
- Approved

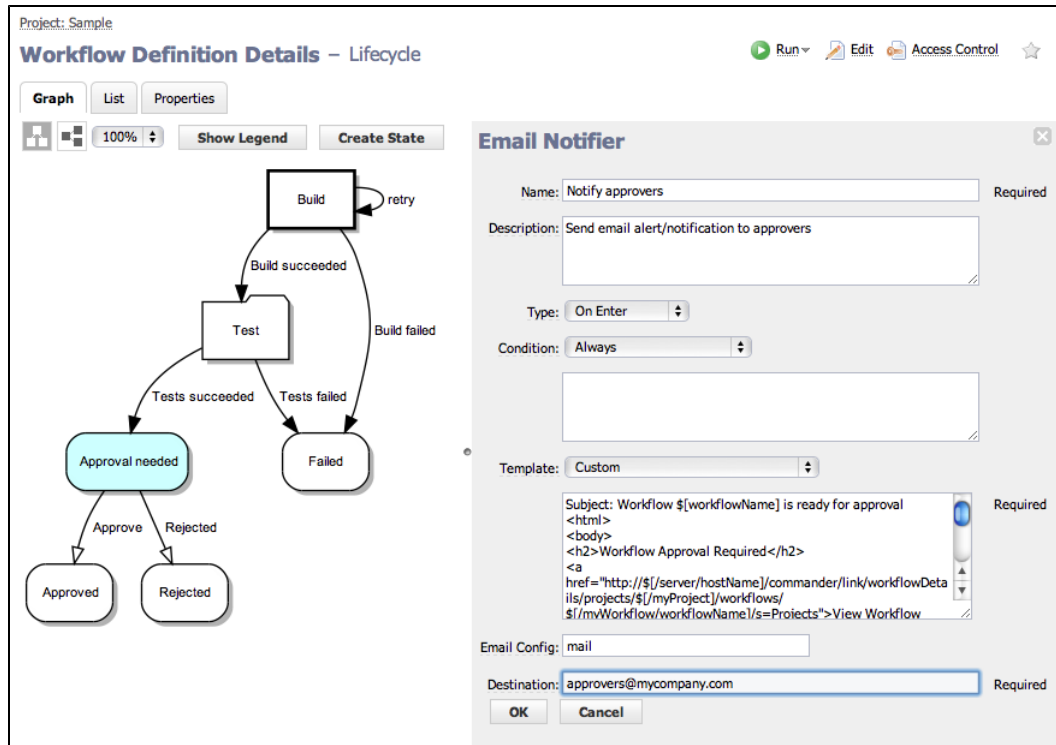
Sending email notifiers

Send email notifications so a user or group knows they need to take an action on a workflow.

- From the Notifications tab for the *Approval needed* state definition, click the Notifiers tab.
- Click the **Create Notifier** link.



- We want to create an On Enter email notifier.
- When a workflow enters the *Approval needed* state, this email notifier will be sent out.



- In this example, we want to send an email notification to the "approvers". The email will contain a link to the Workflow Details page and links to all available transitions.
- Use this formatting template for the email notifier: http://docs.electric-cloud.com/commander_doc/5_1/PDF/WorkflowTutorialScript.txt.

It includes a link to the ElectricCommander workflow, embeds a block of JavaScript that cycles through all the manual transitions from the active state, and has a link to go directly to those transitions. To ensure that the script works properly, use the script as is and make sure there are no white spaces before `<!DOCTYPE...>`, `<html>`, and `</html>`.

- The script above is a sample of the type of script you can insert into the Formatting Template field illustrated in the following screen example.
- When a workflow is run and it reaches the *Approval needed* state, an email similar to the following example can be sent to specified destinations.

Note: Only users who have execute permission on these transitions will be able to take them by clicking the links.



Adding a global parameter to use later in the workflow

Prompt for a parameter when the workflow is launched, and use its value later in the workflow.

- Currently, there is one parameter for the starting state *Build*.
- Click on the *Build* state in the graph and select the Parameters tab.
- Click the **Create Parameter** link to create a second parameter.

Project: Sample

Workflow Definition Details – Lifecycle

Run Edit Access Control

Graph List Properties

100% Show Legend Create State

State Definition

Build [Edit] [View in list] [Create Transition]

Action Parameters Notifiers Properties

Create Parameter

Name	Default Value	Type	Req?	Description	Actions
branch		entry	✓	The branch of the product to build	Delete

- Add a checkbox-type parameter named *skipTests* which, if selected, should skip the test suite after the build is completed.

Project: Sample / Workflow Definition: Lifecycle / State Definition: Build

New Parameter

Name:

Description:

Type:

Value when unchecked:

Value when checked:

Initially checked? ☐

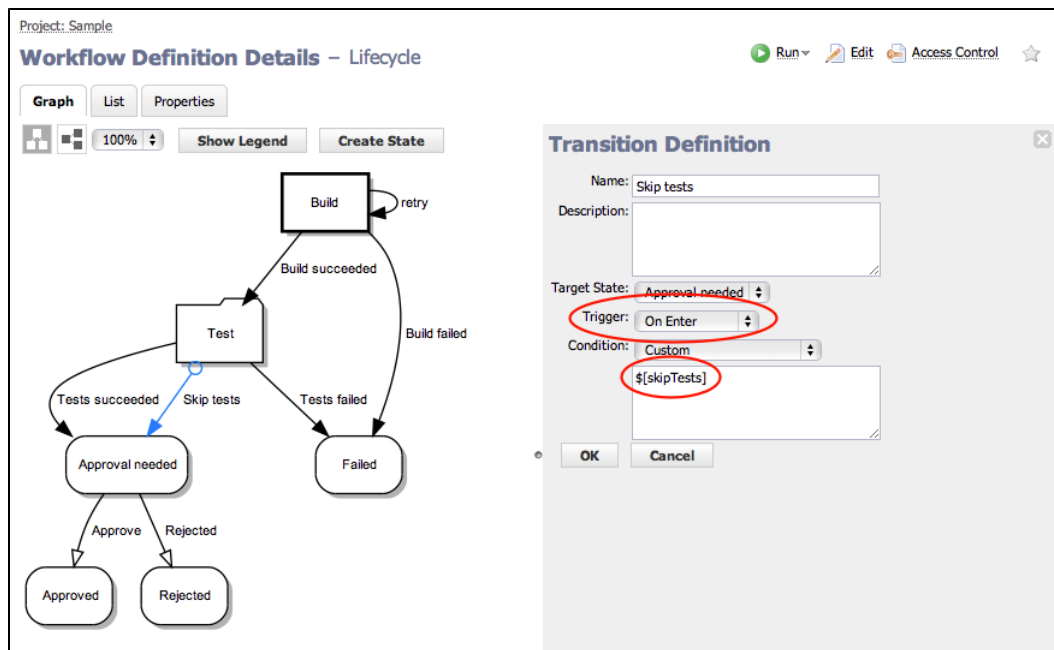
Default Value:

Required? ☐

Defer Expansion? ☐

OK Cancel

- Create an On Enter transition from *Test* to *Approval needed* if the *skipTests* parameter evaluates to "true."
Because this is an On Enter transition, it will be taken before the automated tests workflow is started.

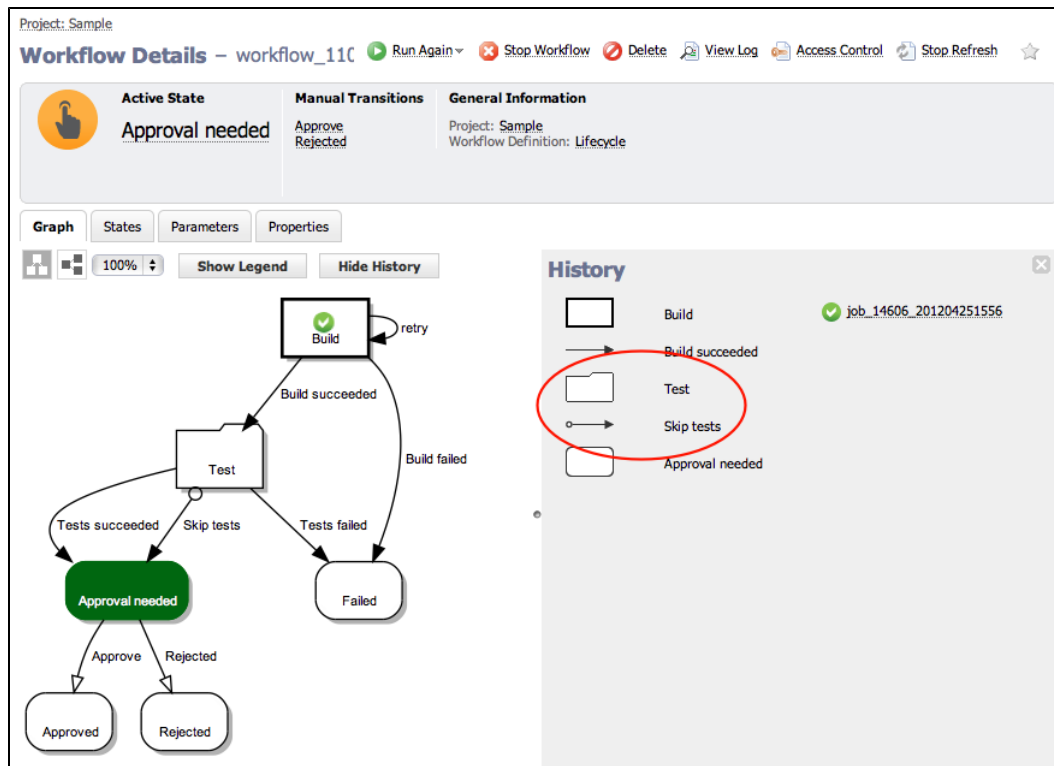


- When the workflow is launched, you will see the *skipTests* checkbox.

The screenshot shows the 'Run Workflow - Lifecycle' interface. It displays 'Starting State: Build' and 'Parameters: branch: 3.5 Required'. Below the parameters, the 'skipTests' checkbox is checked and circled in red. OK and Cancel buttons are at the bottom right.

- If the parameter is checked, and after the *Build* state is completed, the workflow will immediately transition from *Test* to *Approval needed*, without starting the subworkflow.

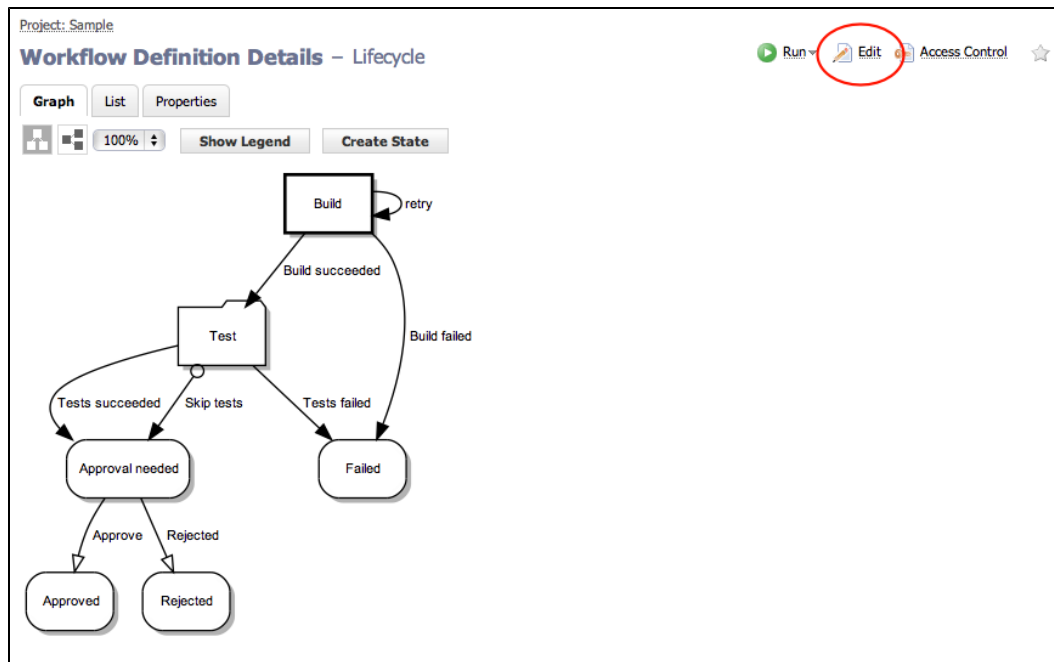
Note: Looking at the workflow History, we see the *Test* state was entered, but the subworkflow (and subjobs) was not executed.



Setting the name of your workflow

Choose appropriate names for workflows so they are easy to identify.

- If the Workflow Name Template was not set, workflows are created with a default name (`workflow <jobId> <timestamp>` or `workflow_239_201012091208`, for example). However, because you may have many different workflow definitions, you might want to create a distinguishing name for workflows launched from each definition.
- From the Workflow Definition page for *Lifecycle*, click the **Edit** link at the top of the page.



- Setting the Workflow Name Template (as shown in the sample below) achieves this goal.
- `lifecycle-${branch}-workflow-${/increment /myProject/lifecycle-counter}`
- `${branch}` refers to the value of the branch parameter as specified when the workflow is launched.
- `${/increment /myProject/lifecycle-counter}` creates a property called `lifecycle-counter` on the project and increments the property every time a new workflow is launched so the names are unique.

Project: Sample

Edit Workflow Definition - Lifecycle

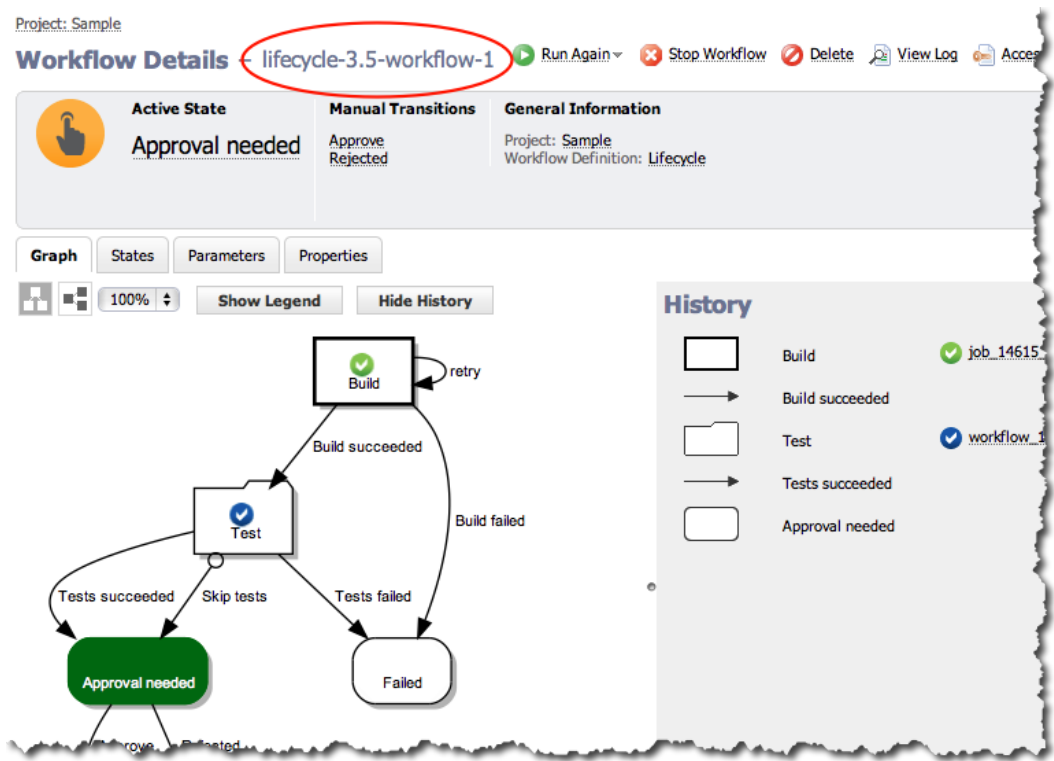
Name: Lifecycle

Description:

Workflow Name Template: `lifecycle-${branch}-workflow-${/increment /myProject/lifecycle-counter}`

OK Cancel

- Now, when the workflow is launched, its name is much more descriptive than the default workflow name.



Workflow List

Most of the time, you will want to interact with your workflow from the default Graph view. However, if you want to see your workflow in a list or table view, click the List button. See the [Workflow Definition Details](#) Help topic for detailed information about this page.

Project: Sample

Workflow Definition Details – Lifecycle

Run Edit Access Control

Graph **List** Properties

Create State Definition

Name	Type	Action	Startable	Description	Actions
Build	Subjob	Sample : BuildProduct	✓		Create Transition Copy Move Delete
retry	On Completion	Target: Build		Retry if the Build state fails	Access Control Copy Move Delete
Build succeeded	On Completion	Target: Test			Access Control Copy Move Delete
Build failed	On Completion	Target: Failed			Access Control Copy Move Delete
Test	Subworkflow	Sample : Automatic Tests			Create Transition Copy Move Delete
Tests succeeded	On Completion	Target: Approval needed		Tests from the sub-workflow have completed with success	Access Control Copy Move Delete
Tests failed	On Completion	Target: Failed			Access Control Copy Move Delete
Skip tests	On Enter	Target: Approval needed			Access Control Copy Move Delete
Failed	No action			An error occurred while the workflow was running	Create Transition Copy Move Delete
Approval needed	No action				Create Transition Copy Move Delete
Approve	Manual	Target: Approved			Access Control Copy Move Delete
Rejected	Manual	Target: Rejected			Access Control Copy Move Delete
Approved	No action				Create Transition Copy Move Delete
Rejected	No action				Create Transition Copy Move Delete

The transition order is important because transitions are evaluated in order and the first one with a condition that evaluates to 'true' is taken. In the next screen example, the *Test* state has two On Completion transitions. When the state completes, it evaluates the *Tests succeeded* transition first. If a third transition was added for the *Test* state, you could use this List view and Move the transition to the correct order for evaluation.

Project: Sample

Workflow Definition Details – Lifecycle

[Edit](#) [Access Control](#) [☆](#)

[Graph](#) **List** [Properties](#)

[Create State Definition](#)

Name	Type	Action	State	Actions
Build	Subjob	Sample : BuildProduct		Create Transition Copy Move Delete
retry	On Completion	Target: Build		Access Control Copy Move Delete
Build succeeded	On Completion	Target: Test		Access Control Copy Move Delete
Build failed	On Completion	Target: Failed		Access Control Copy Move Delete
Test	Subworkflow	Sample : Automatic Tests		Create Transition Copy Move Delete
Tests succeeded	On Completion	Target: Approval needed		Access Control Copy Move Delete
Tests failed	On Completion	Target: Failed		Access Control Copy Move Delete
Skip tests	On Enter	Target: Approval needed		Access Control Copy Move Delete
Failed	No action			Create Transition Copy Move Delete
Approval needed	No action			Create Transition Copy Move Delete
Approve	Manual	Target: Approved		Access Control Copy Move Delete
Rejected	Manual	Target: Rejected		Access Control Copy Move Delete
Approved	No action			Create Transition Copy Move Delete
Rejected	No action			Create Transition Copy Move Delete

Another workflow example

An advanced sample Workflow Definition is pre-defined for you in the EC-Examples project. This is a "working" example, also modeling the workflow process, with which you can interact. To navigate to this example: Select the Projects tab > EC-Examples > Workflow Definitions subtab > then select Build Workflow With Approvals.

Workspaces and Disk Space Management

[Defining workspaces](#)

[Using workspaces](#)

[Workspace directory names](#)

[Working directories](#)

[Workspace accessibility](#)

[Local workspaces](#)

[Access control](#)

[Impersonation and workspaces](#)

[Commander managed files](#)

[Disk space management](#)

[ecremotefilecopy](#)

Overview

ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a **job workspace**. (The job workspace defaults to a directory under the workspace directory.)

- A job step can create whatever files it needs within its workspace, and Commander automatically places files in the workspace, such as step logs. Normally, a single workspace is shared by all steps in a job, but it is possible for different steps within a job to use different workspaces.
- The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.
- Workspaces are grouped together, with workspaces for different jobs allocated as children of the same *workspace root*. When you define procedures, you can specify which workspace root(s) to use, then Commander creates a child directory within that workspace root for the job's steps to use.
- The term "workspace" is used at different times to refer to either a job workspace or a workspace root. Depending on the context, it should be easy to discern which "workspace" is being discussed or referenced.

Defining Workspaces

ElectricCommander can handle any number of workspace roots, which are defined by selecting the Cloud > Workspaces tabs on the web interface. To create a workspace root, provide a workspace name and three file paths for jobs to access the workspace:

Workspace name

The workspace name must not have trailing spaces.

When two workspaces have similar names, such as "Server1" and "Server1 " (with an extra space at the end), the database records them as two entries, one without trailing spaces and one with trailing spaces.

If you run a job against one of these workspaces, it may hang and not proceed because ElectricCommander is waiting for a resource, an available workspace, or a step to complete.

UNC path

A UNC path that Windows machines can use to access the workspace via the network, such as

`//server/share/x/y/z.`

You must ensure this path is accessible on any resource where the workspace is used.

Drive path

A Windows path, using a drive letter that refers to the same directory as the UNC path. Before running a job step using the workspace, the Commander agent creates a drive mapping for this path to match the UNC path.

For example, if the UNC path is `//server/share/x/y/z` and the drive path is `N:`, the agent will map `N:` to `//server/share/x/y/z`. If the drive path is `N:/y/z` then the agent will map `N:` to `//server/share/x`.

The directories in the drive path after the drive letter (`y/z` in the preceding example) must match the last directories in the UNC path.

UNIX path

A path used on UNIX machines to access the workspace via the network, typically using an NFS mount.

You must ensure appropriate mounts exist on all resources where the workspace is used.

If your environment contains Windows machines only, you can omit the UNIX path, and if your environment contains UNIX machines only, you can omit the UNC and Drive paths.

During Commander installation, you had the option of defining a workspace. If you defined a workspace, it was named "default" and will be the default workspace for all jobs, as described below.

By default, remote workspaces are accessed with the agent user's credentials. For Windows resources, you can override that user's credential by attaching a credential to the workspace. Using a credential attached to a workspace is useful if the agent machine is not a member of the domain, but the workspace is located on a machine in the domain. An attached credential for a domain user allows the agent to manipulate the workspace as that domain user.

Using Workspaces

The simplest way to use workspaces is to define a single workspace named "default" that is readable and writable on all resources, and never specify any other workspace information. With this approach the default workspace will be used for all steps in all jobs.

However, you can also arrange for different jobs to use different workspaces, or even for different steps within a single job to use different workspaces.

You can specify a workspace name in any of the following Commander objects:

- If you specify a workspace in the definition of a procedure step, it applies to that step.
- If you specify a workspace in the definition of a procedure, it applies to all steps in the procedure.
- If you specify a workspace in the definition of a project, it applies to all steps in all procedures defined in that project.
- If you specify a workspace in the definition of a resource, it applies to all steps assigned to that resource.

If workspaces are defined in several of these places, the workspace for a particular step is chosen in priority order corresponding to the list above: a workspace specified in a step takes priority over all others, followed by a workspace in the procedure, project, and finally resource. If no workspace is specified in any of these locations, the workspace named "default" is used.

Workspace Directory Names

A job workspace directory name, within its workspace root, is derived from the name of the job.

For example, if a job named "job_63" uses a workspace whose root directory is `//ec/workspaces`, the job workspace directory will be `//ec/workspaces/job_63`. Changing the name of a job has no impact on workspace names: the initial name of the job is always used for workspace directory names.

A job is allocated a single job workspace only within a particular workspace root, which will be shared by all steps that specify that root.

For example, suppose a job has 5 steps: 3 steps specify workspace A and 2 steps specify workspace B. The 3 steps that specify A will all share a single directory under A, and the other 2 steps will share a single directory under B.

Working Directories

By default, each job step begins execution with its current working directory set to the top-level directory in the job's workspace. A job step can override this location by specifying an explicit working directory. If a job step specifies a relative path for its working directory, the path is relative to the top-level workspace directory.

Workspace Accessibility

The simplest way to set up Commander is to make every workspace root accessible on every resource. This accessibility mode provides the most flexibility and simplifies Commander system management.

Unfortunately, some environments cannot allow such universal access.

For example, some sites do not provide file sharing between UNIX and Windows machines. In large enterprise environments where Commander is shared among multiple groups, you may want to partition resources among the groups and limit access by each group to the other groups' resources. In the most extreme case, you may not have network file sharing at all. Commander can handle all of these cases.

If a workspace root is not accessible on a particular resource, you should not use that workspace in any step that runs on the resource. If you do, the resource will not be able to create the job workspace directory and the step will stall until the issue is resolved. A workspace root (on a resource) must be both readable and writable to be used for job steps.

Local Workspaces (Disconnected Workspaces)

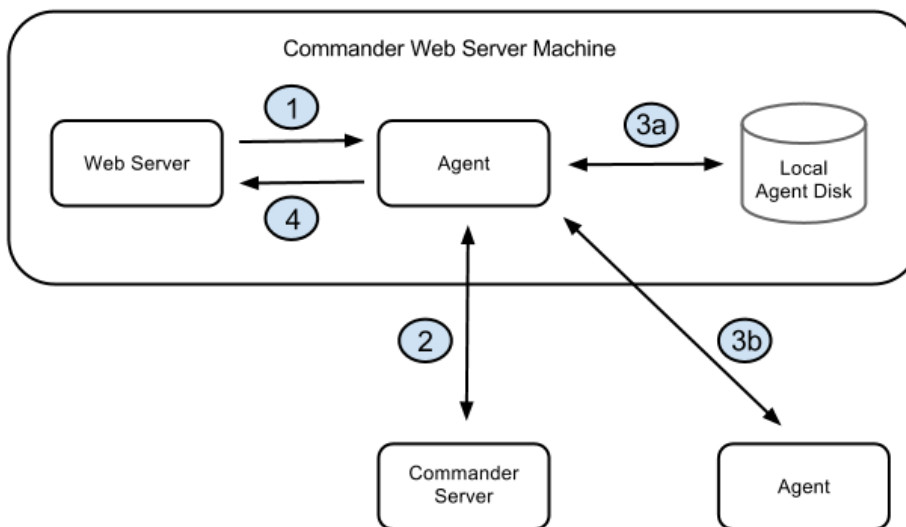
Normally, workspaces are shared, which means if the same workspace is used on different resources, they see the same set of files. However, it is possible to define a *local* workspace (also referred to as a disconnected workspace), which means it will refer to a different local directory on each resource where it is used.

To define a local workspace on Windows, specify a local path such as `C:/Windows/Temp/ecworkspace` for the drive path.

For UNIX, specify a local UNIX path such as `/usr/tmp/ecworkspace`. Local workspaces are advantageous because they do not require remote network access and access may be faster in some cases.

However, you cannot share workspace information among job steps running on different resources.

Note: As of Commander 4.2, web servers "ask" the relevant agent for access to log files in workspaces. The agent streams the file back to the web server, which then streams the log file to your browser. However, this mechanism works only for Windows and Linux 4.2 (or later) agents. Solaris, MacOS, and HP-UX agents must transfer log files to a workspace proactively so a Linux or Windows agent can access them for the web server—this task can be made easier, using the `ecremotefilecopy` tool. See the [ecremotefilecopy](#) section in [Commander Installed Tools](#) for details.



The diagram illustrates the architecture when the web server requests a log file and you have defined a local workspace.

1. The web server asks the local agent to retrieve the file from a workspace.
2. The agent (local to the web server) asks the Commander server how to find the workspace, and the Commander server replies with a route to an agent and validates the session ID.
3. The route to the agent that has the file could be: (a) the local agent itself or (b) a different agent that ran the step that produced the file.
 - a. The agent (local to the web server) gets the file from its local disk.

or

- b. The agent gets the file from some other agent.
- 4. The agent (local to the web server) sends the file to the web server.

Access control

You can restrict access to workspaces using the Commander access control mechanism. Before a step can use a workspace, it must have the "execute" privilege on the workspace, where "it" means the Commander user ID under which the step executes (the project principal). You can set permissions on a workspace to enable or disable access as you choose.

Impersonation and workspaces

Before a job step can use a particular workspace, three access limitations must be satisfied:

The job step must have execute access to the workspace object.

If not, ElectricCommander aborts the step with an access control violation. See the [Access Control](#) Help topic for details.

The workspace must be accessible on the resource where the step runs.

If not, the Commander agent is unable to create the step's log file and the step's execution will fail. See [Workspace Accessibility](#) (above) for more information.

The Windows or UNIX agent account and/or job step must have read/write access to workspace files.

If not, the step's execution will fail in one of several ways. This issue is the topic of this section.

Potentially, two system accounts can impact each job step execution:

The first account is the one used by the Commander agent—you selected this account when Commander was installed. The agent account must have the ability to write the workspace root directory (to create the job workspace directory), and it must have the ability to write the job workspace directory to create log files.

The second account is the one under which the step executes. If the step does not use user impersonation (in other words, no credential has been specified for it), the step runs under the same account as the agent so there are no additional issues. However, if the step is running with credentials, the credentialed account must also have read/write access to the workspace area.

Because of these requirements, you may end up with a configuration where any job step running in a workspace can read or write any file under that workspace root, including files from other jobs that share the same workspace root. In many environments this solution works, but in some situations you may want to prevent one job from accessing files from a different job.

To create job privacy, run all of the job's steps with credentials for a separate account. In addition, run a job step at the beginning of the job that creates a subdirectory within the job workspace and change the protection on that directory to permit access to the credentialed account only. Next, place all private files for the build in the protected subdirectory. These files will not be accessible to other jobs or the agent. The top-level directory in the job workspace still needs to be accessible to the agent so it can read and write log files, but everything in the lower-level directory will be private.

Commander managed files

Commander automatically places certain files in the top-level job step workspace directory:

Step logs

When a step's command executes, its standard output is redirected to a log file unique to that step. The log filename is derived from the step name and its unique identifier within ElectricCommander.

For example, a step named "step1" will have a log file with a name like "step1.2770.log", where 2770 is a unique identifier assigned by ElectricCommander. A unique identifier is needed in situations where the same step name occurs multiple times in a job such as multiple invocations of a single nested procedure.

Postprocessor logs

If a step specifies a postprocessor, standard postprocessor output is redirected to a file in the workspace. The postprocessor output file will have a name similar to the step log. For example, "step1.2770-postp.log".

Diagnostic extracts

If a postprocessor extracts diagnostic information from a step's log file, it usually places that information in an XML file in the top-level job workspace directory, and then it sets a property that contains the filename. The ElectricCommander *postp* postprocessor uses filenames like diag-2770.xml, where 2770 is the unique identifier for the step. Other postprocessors may use different filenames.

Disk space management

The current Commander version does not provide a facility for automatically deleting old workspaces. You are responsible for deleting obsolete job workspaces yourself. If you delete a job, Commander does not automatically delete its workspace. (A future Commander version may provide automated facilities for identifying and deleting obsolete workspaces—until then, you may wish to create a procedure that runs on a regular schedule and deletes old job workspaces.)

Properties Changed to UUIDs

Starting in Commander 5.0, ElectricCommander uses a universally unique identifier (UUID) for the `jobId` and `workflowId` properties. In earlier releases, these properties were monotonically incrementing integers that were created directly by the database. The auto-generated UUID is displayed as an 36-character string. This UUID is stored internally in the database as a 128-bit binary type.

For example:

- A new job in Commander 4.2 may show 1809 as a `jobId`, but launching this same job in Commander 5.0 may result in a `jobId` like this: 4fa765dd-73f1-11e3-b67e-b0a420524153.
- When you migrate a `jobId` from Commander 4.2.x to Commander 5.0, ElectricCommander converts the `jobId` to hexadecimal and adds zeroes before it to create a 36-character string.

If you migrate a Commander 4.2 job with the `jobId` 42978, the `jobId` becomes 00000000-0000-0000-0000-0000000e7e2 in Commander 5.0.

You most commonly reference `jobId` and `workflowId` in the Job name template and Workflow name template fields in these locations:

- Administration > Server > Settings
- Edit procedure or Edit workflow definition

You can use the new Commander 5.0 project-level job counters to count and identify individual jobs.

Note: All entity-table IDs have become UUIDs in Commander 5.0. This topic, however, discusses only `jobId` and `workflowId` because they are the most widely public referenced IDs.

Reasons for the Change

Commander 5.0 changed IDs to UUIDs for several reasons:

- Commander 5.0 allows you to configure a cluster of Commander servers, which adds horizontal scalability and high availability to your Commander implementation. Using UUIDs for all primary keys enables multiple Commander servers to run against the same database without having to coordinate primary key generation across the servers or through the database, which could cause a performance bottleneck.
- In Commander 4.2, the database generated IDs directly. This would create confusion if you moved from one database to another, such as moving from MySQL to Oracle, which would reset the IDs after you uploaded the new database. Using custom counters avoids this ID reset problem and is convenient when moving to a production database such as MS SQL Server or Oracle.

Impact for Commander 4.2 Users

Dependence on `jobId` or `workflowId`

Many customer implementations have relied on job and workflow names that depend on `_${jobId}` or `_${workflowId}`. Although jobs and workflows with these system-generated identifiers do not fail after upgrading to Commander 5.0 or later, the readability of this information will become more difficult for users because previously sequential numbers are replaced by 36-character random hexadecimal UUIDs.

Order of Preference for Name Template Values

Whenever a job is run, Commander names the job according to the job name template value in these locations in descending order of preference:

1. The procedure
2. The property sheet in Administration > Server > Settings
3. The server's hard-coded default

Job Name Templates

Upgrading to Commander 5.0 automatically updates Commander hard-coded default job and workflow name templates. However, you may not see this change in your job or workflow names.

Commander 5.0 *does not automatically update* your name templates if you have ever saved your Commander server settings. It does not matter if you made any changes; the only action that matters is saving the server settings. Saving the server settings creates a custom property sheet, even though it might contain all default values. The creation of the property sheet means job and workflow names are derived from the property sheet's

name templates and not the hard-coded name templates. The hard-coded name templates were updated in Commander 5.0, but because your job and workflow names are not derived from hard-coded name templates, their names are not updated automatically the next time you run a job or workflow. Similarly, your name templates remain as they were in Commander 4.2 if they are located in an individual procedure. You need to run the migration tool to update occurrences of `$_[jobId]` or `$_[workflowId]`.

Commander 5.0 *automatically updates* your name templates if you have never saved your Commander server settings. Because you have never saved the server settings, a custom property sheet does not exist. This means that Commander uses the default hard-coded name templates, which were updated in Commander 5.0. Your job or workflow names automatically use the new default name templates the next time you run a job or workflow.

To determine which name templates Commander is using, follow these steps:

1. In the Commander web interface go to Administration > Server.
2. Check if more than one property is listed in the Custom Server Properties section.

If there are multiple properties listed, you are using a property sheet's templates, not hard-coded templates.

Default Job Name Template Impact

The Commander 4.2 default job name template contains the jobId, as seen in:

```
job_$_[myJob/jobId]_$_[timestamp]
```

If you are using a property sheet's default job name template and you leave the default job name template as-is, job names for new jobs on the ElectricCommander 5.0 server will display with a UUID entry instead of the sequential counter.

If you are using the hard-coded default job name template and you leave the default job name template as-is, your job names will have a job counter instead of the job ID number after you upgrade.

Example 1

You are using the property sheet's default job name template in Commander 4.2. This means you do not have a procedure-level job name template. You do not change anything after upgrading to Commander 5.0.

Assume you have previously run 2500 jobs. In Commander 4.2 your next job name might be

```
job_2501_201401021303
```

In Commander 5.0 that job name would now look like

```
job_4fa765dd-73f1-11e3-b67e-b0a420524153_201401021303
```

Example 2

You are using the hard-coded default job name template in Commander 4.2. This means you do not have a custom property sheet nor a procedure-level job name template. You do not change anything after upgrading to Commander 5.0.

Assume you have previously run 2500 jobs. In Commander 4.2, your next job name might be

```
job_2501_201401021303
```

In Commander 5.0 that job name would now look like

```
Default_2501_20140102130343
```

Customized Job Name Template Impact

If you customized your job name template but still used the jobId, the UUID now appears instead of the previous jobId. This is also true if you used jobId in a procedure-level job name template.

We strongly recommend running the migration tool.

Workflow Name Template Impact

The Commander 4.2 default workflow name template contains the workflowId, as shown in:

```
workflow_${/myWorkflow/workflowId}_${/timestamp}
```

The impact to your workflow names would be the same as what is shown in the job name example.

Actions to Take

- We strongly recommend that you convert your naming templates to reference an incrementing job or workflow counter that is tied to the project.

Examples of naming templates containing properties that increment atomically:

```
${/projectName}_${/increment /myproject/jobCounter}_${/timestamp yyyMMddHHmmss}
```

```
${/projectName}_${/increment /myproject/workflowCounter}_${/timestamp yyyMMddHHmmss}
```

Note: This section references jobId to jobCounter changes, but also consider the same for workflowID to workflowCounter changes.

- If you have any scripts that refer to object IDs and assume that they are numeric, you must update these scripts to handle the IDs as strings.

For example, a Perl script with `my $jobId = ${/jobId};` should be modified by quoting the job ID as follows: `my $jobId = '${/jobId}';`

Migration Tool

Electric Cloud provides the migration tool to ease the transition from jobId and jobCounter.

This script does the following:

- Examines every procedure in your system and makes the following changes to any jobNameTemplate it finds
 - `${/jobId}` becomes `${/increment /myProject/jobCounter}` or `${/increment /server/ec_counters/jobCounter}`
 - Within `${/javascript myJob.jobId}`, `myJob.jobId` becomes `getProperty('/increment /myJob/project/jobCounter')` or `getProperty('/increment /server/ec_counters/jobCounter')`

- Examines every procedure in your system and makes the following changes to any workflowNameTemplate it finds
 - ``${workflowId}`` becomes ``${/increment /myProject/workflowCounter}`` or ``${/increment /server/ec_counters/workflowCounter}``
 - Within ``${/javascript myWorkflow.workflowbId}``, `myWorkflow.workflowbId` becomes `getProperty('/increment /myWorkflow/project/workflowCounter')` or `getProperty('/increment /server/ec_counters/workflowCounter')`
- Determines the current value of the global job ID counter by querying the most recent job. This is used as the starting value for a jobCounter property (you can override this value).

By default, the migration tool runs in *dry run* mode. Dry run mode lists the changes the script would make (showing the *before* and *after* values). You must run the script with an explicit option to actually make the changes. [Using the Migration Tool](#) contains detailed instructions and explanations. Be sure to read that information before you run the migration tool. [jobCounter Considerations](#) contains important information for defining a starting jobCounter value.

You can run this script either before or after you upgrade to Commander 5.0.

Rather than waiting to upgrade to a Commander 5.0 version before using jobCounters, We recommend that you begin exploring the use of jobCounters in your existing production environment. Changing your existing projects to reference a counter rather than jobId can be implemented at any time and allows you to start recording your project-level job-count metrics. Applying these changes to your system before upgrading to the Commander 5.0 release allows you to make these changes at your own pace and avoid concerns being raised by users encountering UUIDs in their job/workflow names after the upgrade.

Default Job Name and Workflow Name Template Changes

Users who upgraded from Commander 4.2 may notice that Commander updated the hard-coded job name and workflow name templates for Commander 5.0. They no longer contain jobId or workflowId.

The new default name templates are:

```
`${projectName}_${/increment /myproject/jobCounter}_${/timestamp yyyMMddHHmmss}`
`${projectName}_${/increment /myproject/workflowCounter}_${/timestamp yyyMMddHHmmss}`
```

Changing from a system jobId to a project-based jobCounter allows you to recognize the number of jobs that each project launches. The previous default of using a system jobId provided only a single system-level metric.

Disadvantages of Using jobId and workflowId

We strongly recommend using jobCounter and workflowCounter instead of jobId and workflowId. These are a few reasons to use jobCounter and workflowCounter instead of jobId and workflowId:

- A migration tool is provided with the install package to automate and ease the transition to jobCounter and workflowCounter.
- jobId and workflowId are no longer human-readable integers so they do not provide any identifiable information and cannot be used as counters.
- Because of the increased length of jobId and workflowId, you might encounter limitations such as maximum Windows path name length, maximum URL length, and maximum command line length.
- The default job name and workflow name templates no longer contain jobId and workflowId.

Using the Migration Tool

The migration tool is an ec-perl script, `migrateTemplates.pl`, which converts `jobNameTemplate` and `workflowNameTemplate` properties containing `$(jobId)` and `$(workflowId)` to use a counter instead.

See [Properties Changed to UUIDs](#) for more information about the change to UUIDs that occurred in Commander 5.0.

Advantages of using jobCounters

Many customer implementations have relied on Job and Workflow names that depend on `$(jobId)` or `$(workflowId)`. Although jobs and workflows with these system-generated identifiers do not fail after upgrading to Commander 5.0 or later, the readability of this information becomes more difficult for users because previously sequential numbers are replaced by 36-character random hexadecimal UUIDs.

You can run the migration tool either before or after you upgrade to Commander 5.0.

We recommend that you begin exploring the use of jobCounters in your existing production environment rather than waiting to upgrade to a Commander 5.x version. Changing your existing projects to reference a counter rather than `jobId` can be implemented at any time and allows you to start recording your project-level job-count metrics. Applying these changes to your system before upgrading to Commander 5.0 allows you to make these changes at your own pace and avoid concerns raised by users encountering UUIDs in their job and workflow names after the upgrade.

Note: This section references `jobId` to `jobCounter` changes, but consider the same for `workflowId` to `workflowCounter` changes as well.

What the Script Does

This script does the following:

- Examines every procedure in your system and makes the following changes to any `jobNameTemplate` it finds
 - `$(jobId)` becomes `$(/increment /myProject/jobCounter)` or `$(/increment /server/ec_counters/jobCounter)`
 - Within `$(/javascript myJob.jobId)`, `myJob.jobId` becomes `getProperty('/increment /myJob/project/jobCounter')` or `getProperty('/increment /server/ec_counters/jobCounter')`
- Examines every procedure in your system and makes the following changes to any `workflowNameTemplate` it finds
 - `$(workflowId)` becomes `$(/increment /myProject/workflowCounter)` or `$(/increment /server/ec_counters/workflowCounter)`
 - Within `$(/javascript myWorkflow.workflowbId)`, `myWorkflow.workflowbId` becomes `getProperty('/increment /myWorkflow/project/workflowCounter')` or `getProperty('/increment /server/ec_counters/workflowCounter')`
- Determines the current value of the global job ID counter by querying the most recent job. This is used as the starting value for a `jobCounter` property (you can override this value).

Running the Migration Tool

By default, the migration tool runs in *dry run* mode. Dry run mode lists the changes the script would make (showing *before* and *after* values). Before you make the actual changes, make sure you examine the proposed changes thoroughly to avoid job name collision.

After you download the upgrade tool, upgrade-bundle.zip, extract migrateTemplates.pl and put it in the bin subdirectory.

To invoke the migration tool:

```
ec-perl migrateTemplates.pl [option] ...
```

The following options are available:

Option	Description
<code>--server <server></code>	Specifies the Commander server to migrate. Default: localhost.
<code>--dryRun <1 0></code>	1 shows prospective changes; 0 actually makes them. Default: 1.
<code>--project <project></code>	Limits migration to the specified project.
<code>--counters <server project></code>	Whether job counters are server-wide or project-specific. Default: project.
<code>--startingJobId <n></code>	Initializes the job counter to the specified number. Default: the current greatest job ID.

Example 1

You want a unique job counter across the entire Commander system and want the next job number to be 1001:

```
ec-perl migrateTemplates.pl --counters server --startingJobId 1000
```

The above example will change `$(jobId)` to `$(/increment /server/jobCounter)` and initialize the job counter to 1000. This means the next job run on the server will be number 1001. The following actually makes the proposed changes:

```
ec-perl migrateTemplates.pl --counters server --startingJobId 1000 --dryRun 0
```

Example 2

You want a unique job counter for each project and want the counters to continue from the current greatest job ID.

```
ec-perl migrateTemplates.pl --counters project
```

The above example will change `$(jobId)` to `$(/increment /myJob/project/jobCounter)` and use the current greatest job ID as the initial value for the job counter.

The following actually makes the proposed changes:

```
ec-perl migrateTemplates.pl --counters project --dryRun 0
```

jobCounter Considerations

When using the migration tool to define a starting jobCounter value, keep in mind the following scenarios and their respective actions:

My job names will keep the same general format, replacing jobld with a new jobCounter

You must set your starting counter at a number higher than the most recent job that was used for this project.

However, you might find it more useful to use a more recognizable base point to allow you to differentiate jobs that began after making this change

Example

The last job run under this project as job 32348.

Technically, setting your counter to “32349” would avoid any possible job name conflicts.

However, it would be probably be more effective to set your counter to “40000”, “100000” or “1000000” to help clarify that any new jobs were jobs that were launched after making this change.

My job names will use a completely different format than in the past and will rely on a new jobCounter

You might simply want to set your jobCounter to “1”.

My job names did not rely on jobld previously; they relied on timestamps for unique naming

No change is necessary, but you might want to consider using a jobCounter in the future to help capture the relative number of jobs occurring under each project.

My job names already use a project-based jobCounter property

No changes are necessary.

My job names already use a server-based jobCounter property

Although you do not have to change this, be aware that using a project-based property instead might provide slight performance improvements during large bursts of job launches.

Job Name Template and Workflow Name Template Best Practices

The most important thing to remember: **make sure each name is universally unique.**

Using a job counter or workflow counter in combination with additional properties will ensure names are unique across your Commander system.

By default, Commander 5.0 and later name templates are set at the project level and they include the project name, for example:

```
${projectName}_${/increment /myproject/jobCounter}_${/timestamp}
```

If you choose to set the template at the procedure level instead, make sure to include the procedure name, for example:

```
${projectName}_${procedureName}_${/increment /myproject/jobCounter}_${/timestamp}
```

Actions that Will Affect the Counter

Following the best practices outlined above will ensure system-wide unique names, but particular actions may cause the counter to behave differently than anticipated.

If you copy a project or procedure, make sure you adjust the counter.

If you do an XML import, run some jobs, and then decide to re-import the same XML source as before, make sure you adjust the counter to the greatest job number before you run more jobs. If you don't adjust the counter, it starts from the most recent import's greatest job number, which means the jobs you ran after the first import but before the second import will not have been counted. This would result in duplicated job numbers but distinct job names if `$/timestamp` were included in the job name template.

Example

Assume the following sequence of events:

1. You do an XML import that has a greatest job number of 3200.
2. Then you run 500 jobs.

The greatest job number is now 3700.

3. Later you decide to re-import the same XML as before.

This makes the greatest job number 3200 again.

If you adjust the job counter to 3700, your next job number will be 3701 and the job name might look like this: `Default_3701_20140115090745`.

Another suggestion is to adjust the job counter to something like 10000, which would clarify which jobs were run after the most recent XML import. The next job name might look like this: `Default_10001_20140115090745`.

If you do not adjust the job counter at all, your next job number will be 3201 (again) and the job name might look like this: `Default_3201_20140115090745`. The previous job number 3201 (after the first import) may have had a job name like this: `Default_3201_20140112153409`. The only difference between the two is the timestamp. If you did not include the timestamp in the job name template and included only the project name and job counter, both job names would be `Default_3201`.

Commander 5.0 Name Template Defaults

The new default job name template:

```
${projectName}_${/increment /myproject/jobCounter}_${/timestamp}
```

The above job name template would produce a job name for the first job run in the "Default" project that would look like this: `Default_1_20140110154958`

The new default workflow name template:

```
${projectName}_${/increment /myproject/workflowCounter}_${/timestamp}
```

The above workflow name template would produce a workflow name for the first workflow run in the "Default" project that would look like this: Default_1_20140109123022

Commander Installed Tools

This reference topic can acquaint you with some Commander tools you may not know about or provide quick access to information for tools you use already. Select from the links in the table below to go directly to information for that tool.

Tool Name	Description
eccert	A command-line tool used to manage the Electric Commander Certificate Authority (CA) and the certificates configured in Commander Server and Commander Agent installations.
ecconfigure	A command-line tool that can change configuration values for any locally installed Commander server, web, agent, or repository service. ecconfigure is a more user-friendly mechanism for configuring aspects of Commander that would otherwise require manual configuration file updates. ecconfigure actually manipulates relevant service configuration files on your behalf.
ecdaemon	A "wrapper" program that can be used to start another program from a Commander job step—the "started" program will run as a daemon process. The Commander agent uses the facilities of the underlying operating system to make sure the process runs in a separate process group on a UNIX-based system, or outside of the normal "Windows Job" grouping in a Windows system. In either case, the Commander agent does not treat the process as one it should wait for or one it should try to "kill" if Commander needs to abort the step.
ecproxy	A driver script with built-in support for SSH. Every major operation can be overridden by defining a Perl function in the Proxy Customization field on the New Proxy Resource panel, available from the Resources page,
ecremotefilecopy	Remote file copy: When Commander agents (on platforms other than Linux or Windows) run steps that create log files in a workspace the Commander web server cannot access (through Linux or Windows agents), use ecremotefilecopy to recreate job logs so they are visible on those Commander agents, which then enables the web server to retrieve and render those log files.
ClusterTool	A command-line tool that imports your ElectricCommander database configuration information into your ZooKeeper server.

eccert

A command-line tool used to manage the Electric Commander Certificate Authority (CA) and the certificates configured in Commander Server and Commander Agent installations.

Usage

```
eccert [ options ] command [ arg ... ]
```

Commands

`addTrustedServer crt` Add a server CA certificate to the agent's keystore.

`getCRL` Retrieve the contents of the current certificate revocation list.

`initAgent [--local | --remote] [options]`

Initialize the agent keystore with a new public/private key pair.
Generates the agent certificate signing request.
If run on the server host, the certificate will automatically be signed by the server CA, and the CA certificate and the signed agent certificate are installed in the agent's keystore.
If run on a non-server host, the signing request is left in the agent directory. If CA Cert is provided, the CA certificate is installed in the agent's keystore.

`--local`

Use the local server CA to sign the agent certificate.

`--remote`

Connect to a remote Commander server to sign the agent certificate.

`--force`

Replace any existing keystore.

`--cname name`

Use the specified name as the common name (CN) in the agent certificate subject. This is normally the fully qualified domain name used by clients to connect to the agent.

```

--altNames entries
    Use the specified list of entries (comma or space separated) as
    the subjectAlternateNames list in the agent certificate. Simple
    names are interpreted as dns entries. Entries may begin with
    "dns:" or "ip:" to indicate the type (for example,
    "ip:192.168.0.1" or "dns:myHost"). If no entries are
    specified, then reverse DNS is used to look up the registered
    name(s) of the host's IP addresses.

initCA
    Initialize the server CA. Creates a new CA key and certificate.

initServer [ options ]
    Initialize the server keystore. Creates and signs the server certificate.
    Installs the CA certificate and the signed server certificate into the server's
    keystore.

--force
    Replace any existing keystore.

--cname name
    Use the specified name as the common name (CN) in the server
    certificate subject. This is normally the fully qualified domain
    name used by clients to connect to the server.

--altNames entries
    Use the specified list of entries (comma or space separated) as
    the subjectAlternateNames list in the server certificate. Simple
    names are interpreted as dns entries. Entries may begin with
    "dns:" or "ip:" to indicate the type (for example,
    "ip:192.168.0.1" or "dns:myHost"). If no entries are
    specified, then reverse DNS is used to look up the registered
    name(s) of the host's IP addresses.

list [ --agent | --server | --index [ --verbose ]

    Display certificate information for agent and/or server keystores or the CA
    certificate index. If no options are specified, both the agent and server
    keystores are listed.

--agent
    List the contents of the agent keystore.

--server
    List the contents of the server keystore.

--index
    List the contents of the CA issued certificates index.

--verbose
    Display additional details.

```

<code>refreshCRL</code>	Refresh the certificate revocation list from the Commander server.
<code>revoke index</code>	Revoke a previously issued certificate by index.
<code>signCertificate csr crt</code>	Sign the certificate signing request provided in file <code>csr</code> and write the signed result to the file <code>crt</code> . The request is rejected by the CA if there is a matching certificate already in the CA database.
<code>updateAgentCertificate crt</code>	Install a previously signed certificate <code>crt</code> into the agent's keystore.

Server Communication Options

<code>--server host</code>	Address of the ElectricCommander server. Defaults to the value of the <code>COMMANDER_SERVER</code> environment variable. If that does not exist, it defaults to <code>localhost</code> .
<code>--securePort port</code>	HTTPS listener port on the server. Defaults to 8443.

Global Options

<code>--help</code>	Print the help message.
<code>--version</code>	Print the version message.

Examples

Example 1: Configure an agent to talk to any server (untrusted mode)

This example generates a new self-signed certificate for the agent and recreates the keystore with no trusted authorities.

```
$ eccert initAgent -force
```

Generating keys

Generating certificate request

```
cname=myAgent.company.com
```

```
san=dns:myAgent.company.com
```

Example 2: Configure an agent to accept connections only from a single remote Commander server

This example generates a new certificate for the agent that is signed by the remote server's certificate authority and installs the signed certificate and its associated trust chain in the agent's keystore. After this point, the agent will only accept requests from the specified server and will be used as a trusted resource by the server.

```
$ ectool --server myserver login admin pw
$ eccert --server myserver initAgent -remote
```

Generating certificate request

```
cname=myAgent.company.com
```

```
san=dns:myAgent.company.com
```

Asking server 'myserver' to sign certificate

Importing 'CA:myserver.company.com' certificate

Importing 'jetty' certificate

Example 3: Configure a Commander server with additional host names in the certificate

This example regenerates the Commander Server Certificate, the specified common name, and alternate subject names to allow trusted connections with multiple external `dns` names.

```
$ eccert initServer --force --cname "myServer.company.com" --altNames "myServer,server2.company.com"
```

Generating keys

Generating certificate request

```
cname=myserver.company.com
```

```
san=dns:myserver,dns:server2.company.com
```

Signing server certificate

Importing 'CA:myserver.company.com' certificate

Importing 'jetty' certificate

ecconfigure

A command-line tool that can change configuration values for any locally installed Commander server, web, agent, or repository service. `ecconfigure` is a more user-friendly mechanism for configuring aspects of Commander that would otherwise require manual configuration file updates. `ecconfigure` actually manipulates relevant service configuration files on your behalf.

Usage

ecconfigure [options]

Commander agent configuration options

<code>--agentInitMemory=percent</code>	Initial java heap size as a percentage of the total system memory.
<code>--agentKeystorePassword=password</code>	Password used to access the agent's keystore..
<code>--agentMaxMemory=percent</code>	Maximum java heap size as a percentage of the total system memory.
<code>--agentInitMemoryMB=size</code>	Initial java heap size in MB.
<code>--agentMaxMemoryMB=size</code>	Maximum java heap size in MB.
<code>--agentLogFile=path</code>	Path where the agent log file should be written.
<code>--agentLogMaxFiles=max</code>	Maximum number of log files to accrue.
<code>--agentLogMaxSize=max</code>	Maximum size of each log file. The value can be suffixed with a unit (MB, KB, B). Without a unit,the value is interpreted as bytes.
<code>--agentPluginsDirectory=path</code>	The path used by the agent to get to the plugins directory on the Commander server where its resource definition lies.
<code>--agentLocalPort=port</code>	Port used by the Commander agent for HTTP communication on the <code>localhost</code> network interface.
<code>--agentPort=port</code>	Port used by the Commander agent for HTTPS communication on any network interface.
<code>--agentProtocol</code>	Protocol used by the agent.
<code>--agentProxyHost=host</code>	The IP address of the proxy server.
<code>--agentProxyPort=port</code>	The port of the proxy server.
<code>--agentNoProxyHosts=hosts</code>	Comma delimited list of hosts that should be reached directly, bypassing the proxy server.
<code>--agentEnableProxySettings=<0 1></code>	Enable or disable the proxy server configuration. If enabling for the first time, <code>--agentProxyHost</code> and <code>--agentProxyPort</code> must be specified.

Apache web server configuration options

<code>--webHostName=host</code>	The host name of the current machine in the form that users will typically use in their browser to access the web server.
<code>--webHttpPort=port</code>	The HTTP port of the web server.
<code>--webHttpsPort=port</code>	The HTTPS port of the web server.
<code>--webTargetHostName=host</code>	The host name of the Commander server to which the web server points.
<code>--webTargetHttpPort=port</code>	The HTTP port of the Commander server to which the web server points.
<code>--webTargetHttpsPort=port</code>	The HTTPS port of the Commander server to which the web server points.
<code>--webTimeZone=timezone</code>	The Olson TimeZone format (example: America/Los Angeles) for the php web server.
<code>--webPluginsDirectory=path</code>	The path used by the web server to get to the plugins directory on the Commander server to which it points.
<code>--webProxyUrl=url</code>	The IP address and port of the proxy server in the following format: <code>http://<IP_ADDRESS_PROXY>:<PROXY_PORT></code>
<code>--webNoProxyHosts=hosts</code>	Comma-delimited host list that should be reached directly, bypassing the proxy server.
<code>--webEnableProxySettings=<0 1></code>	Enable or disable the proxy server configuration. If enabling for the first time, <code>--webProxyUrl</code> must be specified.
<code>--webDLC=url</code>	The URL to use for downloadable content requests.

Commander Server configuration options

<code>--serverFileTransferPort=port</code>	The file transfer port of the server.
<code>--serverHttpPort=port</code>	The HTTP port of the server.
<code>--serverHttpsPort=port</code>	The HTTPS port of the server.
<code>--serverInitMemory=percent</code>	Initial java heap size as a percentage of the total system memory.

<code>--serverMaxMemory=percent</code>	Maximum java heap size as a percentage of the total system memory.
<code>--serverInitMemoryMB=size</code>	Initial java heap size in MB.
<code>--serverMaxMemoryMB=size</code>	Maximum java heap size in MB.
<code>--serverPasskeyFile=path</code>	Path to the server's passkey file.
<code>--serverProxyHost=host</code>	The IP address for the proxy server.
<code>--serverProxyPort=port</code>	The port for the proxy server.
<code>--serverNoProxyHosts=hosts</code>	Comma-delimited host list that should be reached directly, bypassing the proxy server.
<code>-- serverEnableProxySettings= <0 1></code>	Enable or disable the proxy server configuration. If enabling for the first time, <code>--serverProxyHost</code> and <code>--serverProxyPort</code> must be specified.

Repository Server configuration options

<code>--repositoryPort</code>	The repository server port.
<code>-- repositoryInitMemory=percent</code>	Initial java heap size as a percentage of the total system memory.
<code>-- repositoryMaxMemory=percent</code>	Maximum java heap size as a percentage of the total system memory.
<code>-- repositoryInitMemoryMB=size</code>	Initial java heap size in MB.
<code>-- repositoryMaxMemoryMB=size</code>	Maximum java heap size in MB.
<code>-- repositoryStorageDirectory= path</code>	Path to the repository backing store. The artifact repository will use this directory to store artifacts.
<code>-- repositoryTargetHostName=host</code>	The host name of the Commander server to which the repository server points.
<code>-- repositoryTargetHttpPort=port</code>	The HTTP port of the Commander server to which the repository server points.

<code>--repositoryTargetHttpsPort=port</code>	The HTTPS port of the Commander server to which the repository server points.
<code>--repositoryTargetProtocol=<http https></code>	The protocol the repository server uses to talk to the Commander server.
<code>--repositoryProtocol</code>	The protocol the repository server uses to talk to client applications.
<code>--repositoryProxyHost=host</code>	The IP address for the proxy server.
<code>--repositoryProxyPort=port</code>	The port for the proxy server.
<code>--repositoryNoProxyHosts=hosts</code>	Comma-delimited host list that should be reached directly, bypassing the proxy server.
<code>--repositoryEnableProxySettings=<0 1></code>	Enable or disable the proxy server configuration. If enabling for the first time, <code>--repositoryProxyHost</code> and <code>--repositoryProxyPort</code> must be specified.
<code>--repositoryValidateFromDisk=<0 1></code>	Enable or disable disk validation.

General options

<code>-v, --version</code>	Display version information.
<code>-h, --help</code>	Display this information.

Examples

Setting initial and maximum memory settings

For example, to set the Commander Server initial memory percentage to 21% and the maximum memory percentage to 31%, specify:

```
ecconfigure --serverInitMemory 21 --serverMaxMemory 31
```

If your Commander server, web server, or repository server is deployed behind a proxy server that inhibits certain Internet access, you can use `ecconfigure` to set proxy settings for each server in your installation.

To use the following perl scripts, remove the brackets ("`<`" "`>`"), and replace the bracketed example text with the values you need.

- To set Commander Server proxy settings:

```
ecconfigure --serverProxyHost <IP_ADDRESS_PROXY> --serverProxyPort <PORT>
--serverNoProxyHosts "<HOST1,HOST2>"
```

- To set Repository Server proxy settings:

```
ecconfigure --repositoryProxyHost <IP_ADDRESS_PROXY> --repositoryProxyPort
<PORT>
--repositoryNoProxyHosts "<HOST1,HOST2>"
```

- To set Web Server proxy settings:

```
ecconfigure --webProxyUrl <http://IP_ADDRESS:PORT> --webNoProxyHosts
<HOST1,HOST2,HOST3>
```

Changing the Apache web server port

Run `ecconfigure` with `--webHttpPort` and `--webHttpsPort`.

Your web server port setting will be changed appropriately in `httpd.conf`, `ssl.conf`, and `config.php`.

Configuring the backingstore location for the Artifact Repository

```
ecconfigure --repositoryStorageDirectory <new-path>
```

ecdaemon

`ecdaemon` is a "wrapper" program that can be used to start another program from a Commander job step—the "started" program will run as a daemon process. The Commander agent uses the facilities of the underlying operating system to make sure the process runs in a separate process group on a UNIX-based system, or outside of the normal "Windows Job" grouping in a Windows system. In either case, the Commander agent does not treat the process as one it should wait for or one it should try to "kill" if Commander needs to abort the step.

Use Cases

- `ecdaemon` is useful in the case where you are trying to deploy a "server-style" program in a Commander step. You do not want Commander to wait for that step to complete because it may run continuously, but you do want Commander to start the server program and then continue on to the next step.
- `ecdaemon` is useful if you want to "pre-load" some type of background process

`ecdaemon` launches the command and exits. Optionally, it sets a property in Commander with the `pid` of the program it spawned to make it possible for a later step to "kill" the daemonized program if desired.

For example:

```
ecdaemon c:/install.exe a b c
```

Using ecdaemon

```
$ ecdaemon
```

```
Usage: ecdaemon [options] cmd cmdArg1 cmdArg2 ...
       ecdaemon --version
```

Required:

command

The command to daemonize, followed by its args, if any.

Options:

--pidProperty=prop-path

A property to set with the pid of the daemonized process.

--version

Print the version of this program and exit.

Command-line parsing

`ecdaemon` supports the standard UNIX-style `--` flag to indicate there are no more `ecdaemon` options and all subsequent options should be treated as simple arguments to the command.

This is particularly important for commands that themselves take `--` arguments.

For example:

```
ecdaemon /usr/bin/myserver --config /etc/myserver.conf
```

will not run properly because `ecdaemon` will attempt to parse the `--config` option instead of passing it to the `myserver` program.

The correct way to invoke `ecdaemon` in this case is:

```
ecdaemon -- /usr/bin/myserver --config /etc/myserver.conf
```

If you want to store the daemonized process's pid in a property, do so as follows:

```
ecdaemon --pidProperty /myJob/serverPid -- /usr/bin/myserver --config
/etc/myserver.conf
```

As a daemon process, any output goes to `/dev/null`, therefore no output file is generated.

ec-perl considerations

Use the `perl system()` call to start `ecdaemon`.

`System()` returns an exit status, "backticks" capture and return output that waits for the daemonized command to complete on Windows, and `exec` never returns at all if it is successful.

ecproxy

A driver script with built-in support for SSH. Every major operation can be overridden by defining a Perl function in the Proxy Customization field on the New Proxy Resource panel, available from the Resources page (by specifying which operation this function re-implements. These operations must have certain "signatures" for the driver to invoke them properly—the operations are listed and described below. For more detail, see the SSH implementation in `ecproxy.pl`.

ecproxy Algorithm

`ecproxy` invokes the operations detailed below to perform the following actions:

1. Uploads the command-file to a `workingDirectory` on the proxy target, using the protocol specified in the proxy config. Currently, only SSH is supported.
2. Creates a wrapper `sh` shell script that CD's to `workingDirectory`, sets "COMMANDER_ environment variables" that exist in the proxy agent's environment, and runs the command-file previously uploaded.
3. Uploads the wrapper script to `workingDirectory` on the proxy target.
4. Runs the wrapper script on the proxy target and streams its output to the proxy agent's `stdout`.
5. Deletes the local wrapper `shell` script, the remote wrapper, and remote command-file.
6. Exits using the exit code of running the wrapper script.

ecproxy Operations

getDefaultWorkingDirectory

Description	Computes the default working directory where a command needs to run on the proxy target, if the step is not defined with a working directory.
Arguments	None
Returns	The path to a directory as it would be accessed on the proxy target.
SSH implementation function	Not SSH-specific, so the function is 'getDefaultWorkingDirectory'
Existing implementation	Return \$ENV{COMMANDER_WORKSPACE_UNIX};

Reason to override	<p>If the "Working Directory" field is empty on a step that is going to run on a proxy target, the working directory for the step should be the workspace, just as it would be if the step were running on a non-proxy Commander agent.</p> <p><code>ecproxy</code> is guaranteed to run in the workspace directory on the proxy agent, but it is not guaranteed that the proxy target has the same path to the workspace.</p> <p>For example, the workspace on a Windows proxy agent is not the same as the path a Unix proxy target uses to access the workspace—so the existing implementation of this operation simply returns the UNIX path to the workspace. However, if the proxy target has a different path for accessing the workspace, the existing implementation will give the wrong answer. Thus, a user can provide a different implementation that gives the right answer.</p>
--------------------	--

Note: This operation is applicable only if "Working Directory" is empty, and is used as the working directory on the proxy target in that case for running the command.

getDefaultTargetPort

Description	Computes the default port where the proxy target is listening for this protocol.
Arguments	None
Returns	The default port.
SSH implementation function	<code>ssh_getDefaultTargetPort</code>

Note: This operation is applicable only if the resource definition specifies no port value.

connect

Description	Opens a connection to the proxy target using the desired protocol.
Arguments	<code>host, port</code> (optional)
Returns	"connection-context hash-ref" on successful connection. This context can contain anything other functions can use to perform their tasks (for example, a connection handle).
Example 1	<pre>my \$context = connect('myhost', 22)</pre>
Example 2	<pre>my \$context = connect('myhost')</pre>
SSH implementation function	<code>ssh_connect</code>

Note: Because the port is optional, the implementation of `connect` can default to whatever is reasonable for the protocol. This means the 'Proxy Target Port' need not be specified in the Commander web UI for proxied agents reachable on the default port.

uploadFile

Description	Uploads the given <code>srcFile</code> to the proxy target as <code>tgtFile</code> .
Arguments	<code>context</code> , <code>srcFile</code> (typically simple file-name), <code>tgtFile</code> (typically <code>workingDirectory/file-name</code>)
Returns	Nothing; on failure it does a 'die' with an appropriate error message.
Example	<pre>uploadFile(\$context, 'agent123.tmp', '/opt/work/joe/agent123.tmp')</pre>
SSH implementation function	<code>ssh_uploadFile</code>

generateWrapperScript

Description	Generates the script body that will run the command-file on the proxy-target in the <code>workingDirectory</code> .
Arguments	<code>workingDirectory</code> , <code>cmd</code> , <code>cmdArg1</code> , ..., <code>cmdFileName</code> (just the base-name, no directory)
Returns	A string containing the script to execute on the proxy target.
Example	<pre>generateWrapperScript('/opt/work/joe', 'perl', 'agent123.tmp')</pre>
SSH implementation function	Not SSH-specific, so the function is <code>'generateWrapperScript'</code>
Existing implementation	<code>cd workingDirectory</code> ; set <code>COMMANDER_</code> environment variables; run command-file, properly quoting the command and args.
Reason to override	If the proxy target does not have <code>sh</code> , the wrapper script needs to be written in a language available on the target.

uploadWrapperScript

Description	Uploads the wrapper-script code to the proxy target.
Arguments	<code>context</code> <code>workingDirectory.wrapperScriptBody</code>
Returns	The path to the wrapper-script on the proxy target.
Example	<pre>my \$wrapperFile = uploadWrapperScript(\$context, '/opt/work/joe', 'cd /opt/work/joe; perl agent123.tmp')</pre>
SSH implementation function	3.1, 3.1.1: <code>ssh_uploadWrapperScript</code>
3.1.2 and later	Not SSH-specific anymore, so the function is <code>'uploadWrapperScript'</code>

Notes:

1. This function must generate a uniquely named file that will not conflict with other `ecproxy` invocations that might be occurring in parallel steps. The recommended approach is to generate a file name containing the job-step-id.
2. Depending on the protocol and facilities available in the Perl implementation, you may or may not need to create a local `tmp` file to upload to the proxy target. If you do, record that fact in the context and clean up the local file in the `cleanup` operation. Setting the local wrapper file path in `$context->{wrapperFile}` is recommended because the default `cleanup` operation implementation looks for that string.

generateWrapperInvocationCommand

Description	Generates the command-line for running the wrapper script on the proxy target.
Arguments	<code>remoteWrapperFile</code> (path on proxy target)
Returns	A string containing the command-line for running the wrapper script file on the proxy target.
Example	<pre>my \$wrapperCmdLine = generateWrapperInvocationCommand (\$wrapperFile)</pre>
SSH implementation function	Not SSH-specific, so the function is <code>'generateWrapperInvocationCommand'</code>
Existing implementation	<code>Return "sh \$remoteWrapperFile";</code>
Reason to override	The default implementation of this function returns something like <code>'sh \$wrapperFile'</code> . If the wrapper script is not an <code>sh</code> script, or if you want to pass different arguments to the shell, you must override this function.

runCommand

Description	Runs the given command-line on the proxy target.
Arguments:	<code>context</code> , <code>cmdLine</code>
Returns	exit-code from running the command on the proxy target, <code>undef</code> if the command could not be run for some reason.
Example	<pre>runCommand(\$context, \$wrapperCmdLine)</pre>
SSH implementation function	<code>ssh_runCommand</code>

cleanup

Description	Performs any cleanup task after the command has completed on the proxy target. Typically, it deletes any locally created temp files and uploaded files on the proxy target.
Arguments	<code>context</code> , <code>cmdFile</code> , <code>wrapperFile</code> (both are of the form <code>workingDirectory/file-name</code>)
Returns	1 on success, <code>undef</code> on failure.
Example	<pre>cleanupTarget(\$context, '/opt/work/joe/agent123.tmp', '/opt/work/joe/cmdwrapper.123.tmp')</pre>
SSH implementation function	3.1, 3.1.1: <code>ssh_cleanup</code>
3.1.2 and later	Not SSH-specific anymore, so the function is <code>cleanup</code>

Note: The default implementation deletes the locally created wrapper script file whose path is stored in `$context->{wrapperFile}`, if it exists. Thus, if the `uploadWrapperScript` operation is overridden, it is recommended the overriding function set this attribute—that way, `cleanup` need not be overridden.

ping

Description	A test to see if the proxy target is usable.
Arguments	<code>host</code> , <code>port</code> (optional)
Returns	1 on success, <code>undef</code> on failure.
Example	<pre>ping('myhost', 22)</pre>
SSH implementation function	Not SSH-specific, so the function is <code>ping</code> .
Existing implementation	Opens a socket connection to the proxy target on the desired port.
Reason to override	The existing implementation may be deemed too simple for doing a ping; overriding ping to open a connection and do some protocol-specific handshaking might be more appropriate for some protocols / use cases.

Available Helper Functions

To make proxy customization easier, `ecproxy` provides the following helper functions.

mesg

Description	Debug logging function. Writes to the file referenced in the <code>ECPROXY_DEBUGFILE</code> environment variable (if it exists). No-op otherwise.
Arguments	<code>message</code>

Example `mesg("myCleanup: about to delete $cmdFile on proxy target");`

Note: This function automatically adds a newline to whatever it emits, so the caller does not have to incorporate a newline in message.

readFile

Description	Reads a file.
Arguments	fileName
Returns	Contents of the file. If there is an error, it returns an empty string.
Example	<code>my \$data = readFile("foo.txt");</code>

writeFile

Description	Creates a local file containing data.
Arguments	fileName, data
Returns	1 on success, undef on failure.
Example	<code>writeFile("myWrapper.\$ENV{COMMANDER_JOBSTEPID}.cmd", "perl foo.pl")</code>

initDispatcher

Description	Initialize the operation dispatcher map to point to functions for the given protocol. For each operation, <code>initDispatcher</code> checks if a function named <code>protocol_operation</code> exists, and if so, assigns that function as the implementation for that operation.
Arguments	protocol
Example	<code>initDispatcher("ssh")</code> sets the "connect" operation to run "ssh_connect", "uploadFile" => "ssh_uploadFile", etc.

setOperation

Description	Sets the implementation of an operation to be a particular function.
Arguments	operation, function. The 'function' argument may be the name of a function or a reference to a function.
Example	<code>setOperation("ping", "my_ping");</code> sets the "ping" operation to run the "my_ping" function

Example	<code>setOperation("ping", \&my_ping);</code> same as above, but using a function ref
---------	---

Note: This function manipulates the `gDispatcher` hash, but provides a safe interface to it.

loadFile

Description	Load proxy customizations from a file.
-------------	--

Arguments	<code>fileName</code>
-----------	-----------------------

Example	<code>loadFile("custom.pl")</code>
---------	------------------------------------

setSSHKeyFiles

Description	Set the paths to the public and private key files that ssh will use to authenticate with the proxy target.
-------------	--

Arguments	<code>publicKeyFile, privateKeyFile</code>
-----------	--

Example	<code>setSSHKeyFiles('c:\foo\pub.key', 'c:\foo\priv.key')</code>
---------	--

Note: This is very useful on Windows proxies, where there is no reasonable default for ssh to use.

setSSHUser

Description	Set the name of the user to authenticate with the proxy target.
-------------	---

Arguments	<code>userName</code>
-----------	-----------------------

Example	<code>setSSHUser('user1')</code>
---------	----------------------------------

Note: By default, the user name the agent is "running as" is used to log into the proxy target. If key-based authentication is configured on the target system such that 'agentUser' can log into the 'user1' account on the proxy target, this function leverages that configuration.

useMultipleSSHSessions

Description	Normally, <code>ecproxy</code> uses one ssh session with a number of "channels" to perform tasks like uploading files, running the command, and running a cleanup command on the proxy target. Some SSH servers don't allow this. This method configures <code>ecproxy</code> to use a separate SSH session for each operation; this requires authenticating with the SSH daemon on the proxy target several times, and thus it may perform worse than the single-session-multi-channel mode.
-------------	---

Arguments	None
Example	<code>useMultipleSSHSessions()</code>

Examples

Specify public/private key files for SSH

1. Set the `proxyCustomization` property on the resource like this: `?setSSHKeyFiles ('c:\foo\pub.key', 'c:\foo\priv.key');`
2. Set the `ECPROXY_SSH_PRIVKEYFILE` and `ECPROXY_SSH_PUBKEYFILE` environment variables on the proxy agent as system environment variables.

Override one of the operations (for example, to enable SSH connection with username/password)

Set the `proxyCustomization` property on the resource like this: `?sub myConnect($$) {...}`
`setOperation("connect", \&myConnect);`

Load proxy customizations from a file rather than having all the logic in the `proxyCustomization` property on the resource

Set the `proxyCustomization` property on the resource like this: `?loadFile('c:\foo\custom.pl');`

Implement a whole new protocol

Specify protocol as `'myproto'` and have a proxy customization block like this:

```
sub myproto_getDefaultTargetPort() {
  ...
}
sub myproto_connect($;$) {
  ...
}
sub myproto_uploadFile($$$) {
  ...
}
sub myproto_uploadWrapperScript($$$) {
  # Note: As of 3.1.2, the default implementation is likely good enough, so
  it may not be necessary to define this override
  ....
} sub myproto_runCommand($$) {
  ...
} sub myproto_cleanup($$$) {
  # Note: As of 3.1.2, the default implementation is likely good enough, so
  it may not be necessary to define this override
  ....
}
# Initialize the dispatcher to run these functions
initDispatcher("myproto");
```

Override ping to do a connect operation (which does a full protocol handshake, authentication, and so on)

Write a specialized ping function for the proxy customization like this:

```
sub heavy_ping($$) {  
  my ($host, $port) = @_;  
  return ssh_connect($host, $port);  
}  
setOperation("ping", \&heavy_ping);
```

"Real World" Examples

ClusterExec

A basic integration for using `clusterupload` and `clusterexec` to reach a proxy target is here. It has been tested on a Windows target with a Cygwin installation. It will not work "out-of-the-box" because it makes the following assumptions:

- The proxy target has `sh` and other UNIX tools (for example, `rm`).
- The locations of the `clusterexec` and `clusterupload` binaries are hard-coded at the top of the proxy customization.

To make this proxy customization work on a Windows machine that does not have Cygwin, the `generateWrapperScript` operation would need to be overridden with a function that generates a `cmd` batch script, and the `generateWrapperInvocationCommand` operation would have to be overridden to generate a `"cmd /c ..."` command rather than `"sh ..."`.

MySQL

The idea here is that the proxy target need not be a host for running arbitrary commands. It could be a special entity like a `db`. This integration uses the `mysql` `clt` to run the step command (which should be SQL) on the `db` referenced by the proxy target host and port.

A bare-bones integration with MySQL:

```
# Set the path to the mysql binary; if the directory is in the proxy agent's  
# PATH, this variable can simply contain the name of the executable.  
  
my $gMySQL = "c:/cygwin/usr/local/tools/i686_win32/bin/mysql.exe";  
  
sub mysql_getDefaultTargetPort() {  
  return 3306;  
}  
  
sub mysql_connect($;$) {  
  # This "protocol" implementation is just going to use the mysql  
  # command-line tool, so just save off host/port.  
  
  my $host = $_[0];  
  my $port = $_[1] || mysql_getDefaultTargetPort();  
  
  return {host => $host,  
          port => $port};  
}  
  
sub mysql_uploadFile($$$) {
```

```

my ($context, $cmdFile, $rmtCmdFile) = @_;

# We do not need to upload the command-file to the proxy target.
# We are going to run the mysql clt on the proxy agent to run
# the query (contained in the local command-file),
# so just save off the name of the command-file.

$context->{cmdFile} = $cmdFile;
}
sub mysql_uploadWrapperScript($$$) {
    my ($context, $workingDir, $wrapperScript) = @_;

    # This has no meaning for this integration. No-op.
}
sub mysql_runCommand($$) {
    my ($context, $cmdLine) = @_;

    # cmdLine is a command-line for running the wrapper script, which
    # has no meaning for this integration. We just want to run
    # 'mysql' for the desired host/port with the command-file.

    system("$gMySQL -D commander -h $context->{host} -P $context->{port} " .
        "-u commander -pcommander -e \"source $context->{cmdFile}\"");
}
sub mysql_cleanup($$$) {
    # We didn't create any temp files. No-op.
}
# Initialize the dispatcher to run these functions
initDispatcher("mysql");

```

Android

This example uses the `adb` tool to upload files to the device and run commands on it. Initial testing has been only against the android emulator, but it is implemented in such a way that it should work against a real android device attached using USB to the proxy agent, or a device on the network.

A first attempt at proxying to android devices:

```

# Set the path to the adb binary; if the directory is in the proxy agent's
# PATH, this variable can simply contain the name of the executable.

my $gADB = "c:/android-sdk-windows-1.6_r1/tools/adb.exe";

sub android_getDefaultTargetPort() {
    # Not sure what a good meaningful value is here.
    return 0;
}
sub android_connect($;$) {
    # This "protocol" implementation uses the adb
    # command-line tool. Depending on the value of
    # host, construct the appropriate adb command-line
    # argument.

    my $host = $_[0];
    my $context = {};

```

```
# if ($host eq "emulator") {
  if ($host eq "localhost") {
    # We want to talk to the emulator running on this host.
    $context->{targetArg} = "-e";
  } elseif ($host eq "usb") {
    # We want to talk to the single android device connected
    # to the computer via a USB.
    $context->{targetArg} = "-d";
  } else {
    # This must be the serial number of some device somewhere.
    $context->{targetArg} = "-s $host";
  }
  return $context;
}

sub android_uploadFile($$$) {
  my ($context, $srcFile, $tgtFile) = @_;
  my($filename, $directories) = fileparse($tgtFile);

  my $result = `$gADB $context->{targetArg} push $srcFile
"/data/tmp/$filename" 2>&1`;
  if ($? != 0) {
    die ("android_uploadFile: Error uploading file $srcFile to
/data/tmp/$filename: $result\n");
  }
}

sub android_runCommand($$) {
  my ($context, $cmdLine) = @_;

  # cmdLine is a command-line for running the wrapper script, which
  # has no meaning for this integration. We just want to run
  # 'adb' for the desired device with the command-file.

  system("$gADB $context->{targetArg} shell $cmdLine");
}

sub android_cleanup($$$) {
  my ($context, $remoteCmdFile, $remoteWrapperFile) = @_;

  # This was copied from ssh_cleanup except that we do "rm",
  # not "rf -f".

  msg("cleaning up");

  # Delete the locally generate wrapper file.
  unlink($context->{"wrapperFile"});

  # Delete the cmd-file and wrapper script file on the proxy target.
  $gDispatcher{"runCommand"}($context,
    "rm $remoteWrapperFile $remoteCmdFile");
}

sub android_ping($;$) {
  my ($host, $port) = @_;
  $port = $gDispatcher{"getDefaultTargetPort"}() unless isPortValid($port);

  my $socket = IO::Socket::INET->new(PeerAddr => $host,
                                     PeerPort => $port,
                                     Proto   => "tcp",
```

```

                                Type      => SOCK_STREAM)
                                or die "Couldn't connect to $host:$port : $@\n";
    }

    # Initialize the dispatcher to run these functions
    initDispatcher("android");
    1;

```

ecremotefilecopy

When Commander agents (on platforms other than Linux or Windows) run steps that create log files in a workspace the Commander web server cannot access (through Linux or Windows agents), use *ecremotefilecopy* to recreate job logs so they are visible on those Commander agents, which then enables the web server to retrieve and render those log files.

Using `postp` and *ecremotefilecopy*, the log file is populated and recreated in a workspace accessible to the Commander web server, allowing the Job Details page to display the log file. Although this functionality is supported, it is not a recommended method of operation. This method should be used only as a last resort when a shared file system (between alternate agents and primary platform agents [Linux and Windows]) is not possible.

The reasons *ecremotefilecopy* is not recommended are:

- You will not see logs in real time. Logs are not visible until the "recreate step" has completed running.
- There is a performance penalty, especially when running with large files.

Setting up the process

- Create a "Setup" step in your procedure
- Update the Postprocessor field for each step whose results you want to see on the server.
- Add a step (one or more times) to the procedure to recreate the Commander server log files.

Creating a Setup Step

In your procedure, create a step called "Setup". This step needs to be in your procedure **before** any step running on a remote workspace.

Note: This is your top-level procedure, not a subprocedure.

- Navigate to your procedure.
- To create a new step, click the Command step link.
- Set the fields as follows:

Step Name: Setup

Command(s): `ecremotefilecopy setup`

Resource: local

Workspace: you have two choices:

- `use default`
- `use: alternateWorkspaceForDisplay`
There is a property on the Workspace called `alternateWorkspaceForDisplay`, which is a secondary location to look for workspace files. This secondary location is used when the workspace files are not accessible to the web server. If the Apache server cannot locate the file in the original workspace, it looks in the alternate one.

Update the Postprocessor field for steps in your procedure

This step defines a postprocessor that will run at the end of the steps you specify. Add the following information to every step running on a remote agent if you want to see its results in the web interface.

- Navigate to a procedure and a step.
- In the Postprocessor field, enter:
`postp --check none --loadProperty /myJob/jobSteps[Setup]/postpExtensions`
- If you are using `postp` in this step to scan your step log for errors, warnings, and so on also, omit "`--check none`" from the invocation line.

Add a Final Step to your procedure

This step adds a new step at the end of your existing procedure. This step finds all properties created by the postprocessor, then reads the properties and creates local log files based on the properties, then deletes the properties.

- Navigate to your procedure.
- To create a new step, click the Command step link.
- Set the fields as follows:

Step Name: `Recreate the Log Files`

Always run step: (Check the box)

Command(s): `ecremotefilecopy recreateFiles`

Resource: `local`

Workspace: `default`

After the final step runs, you should see links (icons) displayed in the Log column on the Jobs Details page.

Click the icon to display the log file.

Copying Other Files from the Workspace

By default, `ecremotefilecopy` copies only `postp` log and diag files, and step logs. You can also copy other files from the workspace using a function named `postpEndHook2`.

You must do the following in your step:

1. Make sure that the file you want to copy is in the step workspace. (It can be copied there, created there, etc.)

2. For your procedure, create a property (named `postpEndHook2`, for example).
3. Define a function named `postpEndHook2` inside the property. For example:

```
sub postpEndHook2 () {

    # Missing param does not cause an error
    $::gCommander->abortOnError(0);

    # Add filename to a "special" property such that it will be picked up by ecr
    emotefilecopy
    my $fileName = 'paul.txt';
    copyFileToProperty ($fileName);

    # Restore default error handling
    $::gCommander->abortOnError(1);
}
```

4. Add the following line in the Postprocessor field of this step:

```
postp --check none --loadProperty /myJob/jobSteps[Setup]/postpExtensions --loadP
roperty /myProcedure/postpEndHook2
```

ZKConfigTool

You can ZooKeeper server with configuration information that all ElectricCommander server nodes will use in a clustered configuration. This command-line tool that imports your ElectricCommander database configuration information into your ZooKeeper server. The following minimum set of files is imported:

- `database.properties`
- `keystore`
- `passkey`
- `commander.properties`

Prerequisites

- The ElectricCommander Tools package must be installed on the system.
- The system must be running a version of Java supported by ElectricCommander. Java is automatically installed on a system with the ElectricCommander software as part of the Tools install.
- The ZooKeeper software must be installed on the network.

Command

```
$ java -jar zk-config-tool-jar-with-dependencies.jar -<options>
```

Usage: ZKConfigTool

Option	Description
<code>-commanderPropertiesFile <path_to_file></code>	Import the ElectricCommander <code>commander.properties</code> file.
<code>-databasePropertiesFile <path_to_file></code>	Import the ElectricCommander <code>database.properties</code> file.
<code>-help</code>	Show the command help.
<code>-keystoreFile <path_to_file></code>	Import the ElectricCommander <code>keystore</code> file.
<code>-passkeyFile <path_to_file></code>	Import the ElectricCommander <code>passkey</code> file.
<code>-wrapperConfFile <path_to_file></code>	Import the <code>wrapper.conf</code> file.
<code>-writeFile <path_on_zookeeper> <path_to_file></code>	Write the specified file to the ZooKeeper server.

Import Files

Run the `ClusterTool` command to completely populate the ZooKeeper server with configuration information. The system must have the Tools install and can be able to communicate with ZooKeeper.

- If you run the command with defined file options, the tool immediately attempts to contact the specified ZooKeeper system and import the specified files.

Example command:

```
COMMANDER_ZK_CONNECTION=10.168.33.10:2181 java -jar zk-config-tool-jar-with-  
dependencies.jar --databasePropertiesFile database.properties --keystoreFile  
keystore --passkeyFile passkey --wrapperConfFile wrapper.conf --  
commanderPropertiesFile commander.properties
```

- If you run the command without defined file options, the tool displays a series of prompts for the required Commander information. Once the correct information is entered, the tool contacts the specified ZooKeeper system to import the specified files.

Example command:

```
$ COMMANDER_ZK_CONNECTION=10.168.33.10:2181 java -jar zk-config-tool-jar-with-  
dependencies.jar
```


Web Interface Help

All ElectricCommander help topics above this directory are expanded, user-guide style help topics about a particular Commander feature or function. Direct access to these help topics is from the help system Table of Contents. Although these topics are not directly linked within the Commander product, many "page-specific help topics contain links to one or more of these topics to provide more detailed information quickly.

The help topics listed below this directory are "page-specific" help topics, which means each of these topics contains specific information for the Commander web page you are viewing. To access these help topics while using Commander, click the **Help** link at the top-right corner on any Commander web page. (The topics in this section of the help directory are listed singularly or in groups as is appropriate.)

Access Control

Use this page to view and modify access privileges for a particular ElectricCommander object. Depending on the object where you want to set permissions, you will see that object's name as part of the page title (above the tables).

For example, if you clicked the Access Control link from a Project Details page, you will see the project name as part of the Access Control page title.

Reading and Using This Page

- This page displays one or more access control lists. The top list contains entries for the object itself (specified in the page title) and also identifies the object.
For example, if the heading for the top list reads "Privileges for Procedure: buildAndTestAll," you are viewing access privileges for a procedure named "buildAndTestAll". Click the object name to view the main page for that object.
- Typically, you will see more than one list on the page. Each list below the first one contains privileges for an object that "contains" the objects above it.
For example, a project contains all of its procedures and a procedure contains all of its steps. Privileges for the top-level object are determined by all the privileges in all of the displayed lists. The lists form an "inheritance chain" where each object "inherits" permissions from the objects below it on the page.
- When a user attempts a particular operation on the object, ElectricCommander examines the lists on this page from top to bottom. If there is an entry specifying "deny" for the user (or a group containing the user) in the top list, access is denied. Otherwise, if an entry specifies "allow" for the user (or a group containing the user) in the top list, access is allowed. If access is neither allowed nor denied by the top list, ElectricCommander proceeds to the next list and processes it in the same way.
Note: If access is neither allowed nor denied by any list, Commander denies access.
- The inheritance mechanism makes it easy to control access for a large number of objects in a single place.
For example, project access control entries automatically apply to new objects created within the project. Each new object in the project has an empty access control list, but will inherit from the project.

Using the Links...

Use these links to add or increase Access Control for an object.

- **Add User** - Use this link to add permissions for a specific user.
- **Add Group** - Use this link to add permissions for a specific group, which means all users in that group would have the permissions allowed to the group.
- **Add Project** - Use this link to set or redefine permissions for a project.

- **Break Inheritance - Caution: Be very careful if you break inheritance!** If you use the "Break Inheritance" action for any list, no additional inheritance occurs *below* that list and you no longer see other lists on this page. This action is useful if you want privileges for an object to be totally different than its containing object. If an object has no entries in its access control list and you break inheritance for that object, you make the object *completely inaccessible* - you will not even have the "Change Permissions" privilege, so you cannot restore inheritance. If this occurs, see your system administrator to restore inheritance.
- Actions:
 - **Edit** - Use this link to modify current permissions, but be careful if you modify permissions in an *inherited* access control list. Modifying inherited access control affects all other objects that inherit from the same list.
 - **Delete** - Deletes the current privileges granted for that user, group, or project.

Privilege Definitions

The following four privilege types (for each Commander object) can be assigned *allow*, *deny*, or *inherit* permission.

Read - Allows object contents to be viewed.

Modify - Allows object contents (but not its permissions) to be changed.

Execute - If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.

Change Permissions - Allows object permissions to be modified.

For more information, see the main [Access Control](#) help topic. This help topic also contains two examples that illustrate how you might use Access Control to increase Commander security.

Access Control - defining entries

Use this page to define access control entries selected from a filtered list of Commander users, groups, or projects, or by providing an explicit name.

Fill-in the fields as follows:

Note: The following instructions are generic. For example, if you chose Add User, the Filter field will be labeled "User Filter".

Field Name	Description
Radio selector	Choose between selecting "Search for existing..." or select "Provide explicit ... name". Providing an explicit name bypasses the search and allows you to enter an arbitrary name.
Filter	Enter a partial name to retrieve a list of all users/groups/projects that include the partial name you supplied.
Include Inactive	<p>[User/Group only] - Use this checkbox only if requesting external LDAP or Active Directory providers.</p> <p>If "checked," all available users or groups are returned.</p> <p>If not checked, only those users or groups known to the Commander server are returned, for example, users who have logged in or groups containing users who have logged in.</p>

Click **OK** to retrieve your information.

If you entered filter criteria, the page will refresh with a list of matches. Select any row in this table to create an access control entry for that principal.

Column descriptions

Name - The name of the user/group/project.

Location - Only viewable for users or groups from non-local directory providers. The name combined with the location is the unique identifier for this user or group, local to the Commander server.

Privileges - create new or edit existing privileges

Use this page to create or modify an access control list entry. Each entry names a *principal*, which can be either an individual user or a group, and defines four privileges for the principal.

For each of the four privileges, **Read**, **Modify**, **Execute**, or **Change** Permissions:

- Creating a new entry or modifying an existing entry allows you to select either Inherit, Allow, or Deny access entries.
- On the server access control table, "Don't Care" replaces the "Inherit" option.

For more information about access control, see the [Access Control](#) Help topic.

Artifacts

This page displays all artifacts available on this Commander server.

Links and actions at the top of the table

- **New Search** - Use this link to go to the Define Search page to filter the Artifacts page view.
- **Create Artifact** - Use this link to go to the New Artifact page to create a new artifact.
- The "star" icon allows you to save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Name	The name of the artifact—a system-generated name created by combining the Group Id and Artifact Key components. Note: Selecting an artifact name takes you to the Artifact Details page for that artifact.
Group Id	The user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
Artifact Key	The "key" component of the artifact name.
Description	The plain text or HTML description previously supplied for this artifact.
Actions	Edit - Use this link to go to the Edit Artifact page to modify the artifact on this row. Delete - Use this link to delete the artifact on this row.

Artifact Details




This page displays detailed information about an artifact.

Links and actions at the top of the table

- **Edit** - Use this link to go to the Edit Artifact page to make any necessary changes for this artifact's description or artifact version name template.
- **Access Control** - Use this link to set privileges for this artifact. *For more information*, see the [Access Control](#) Help topic.
- The "star" icon allows you to save this job information to your Home page.

General Information section

This section displays the artifact name (Artifact), Group Id, Artifact Key, and any Description previously supplied for this artifact.

Artifact Details - exports:project   

General Information
Artifact: exports:project
Group Id: exports
Artifact Key: project
Description:

The "tabbed" sections

The next section of tabs allows you to select the type of information you want to see.

Artifact Versions table

This table displays all artifact versions included in this artifact.

Artifact Versions		Properties
Artifact Version	Description	Actions
exports:project:1.0.0-2149		Edit Delete
exports:project:1.0.0-2167		Edit Delete

Column descriptions

Artifact Version - Click on an artifact version name to go to the Artifact Version Details page for that artifact version.

Description - The previously supplied description, if any, for this artifact version.

Actions

- Click the **Edit** link to go to the Edit Artifact Version page.
- Click the **Delete** link to delete this artifact version from the artifact.

Properties table

This table contains custom properties for this artifact.

- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Sheet**, and **Access Control** links to add more custom properties or change or set privileges on this artifact.

Artifact Versions		Properties	
		Create Property Create Nested Sheet Access Control	
Property Name	Value	Description	Actions
prop1start	1		Delete
prop2add			Delete
prop3export	complete		Delete

Artifact - create new or edit existing artifact

To create a new artifact

Fill-in the fields as follows:

Field Name	Description
Group Id	Supply a group name of your choice for this artifact. A Group Id (groupId) acts as a namespace for grouping related artifacts. The idea is that artifact "sdk" for group "platform" can co-exist with artifact "sdk" in group "ui" without there being any name collisions. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
Artifact Key	Supply an identifier of your choice for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
Artifact Version Name Template	<p>A template for the names of artifact versions published to this artifact.</p> <p>For example:</p> <pre>\$/myArtifactVersion/groupId]:\$/myArtifactVersion/artifactKey]:\$ [/myArtifactVersion/version]</pre> <p>Produces a name like: platform:sdk:1.0.0-36</p> <p>Only absolute property paths and /myArtifactVersion-based paths are supported. If not specified, the "Artifact Version name template" server setting is used.</p>

Click **OK** after filling-in the fields and to go to the Artifacts page to see your new artifact listed in the first column.

To edit an artifact

- Click the **Access Control** link at the top of the page to add permissions to this artifact.
- You can add a Description or modify your existing description.

- You can add an Artifact Version Name Template or modify your existing template.
- Click **OK** to save your changes and return to the Artifacts page.

Artifact Versions

This page displays all artifact versions available on this Commander server.

Links and actions at the top of the table

- **New Search** - Use this link to go to the Define Search page to locate artifacts, repositories or other related information.
- The "star" icon allows you to save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Name	The name of the artifact version. Select an artifact version name to go to the Artifact Version Details page for that artifact version.
Artifact	The name of the artifact. Select an artifact name to go to the Artifact Details page for that artifact.
Group Id	The name of the group where this artifact version is a member.
Artifact Key	The "key" identifier for this artifact version.
Version	The artifact version represented, by default, as <code>major.minor.patch-qualifier-buildNumber</code> . Note: You will not see all 5 artifact version components specified if only 3 components were defined.
State	The current state of this artifact version. Possible values are: <code>available publishing unavailable</code> .
Modify Time	The last time this artifact version was modified.
Actions	Edit - Use this link to go to the Edit Artifact Version page to modify the artifact version on this row. Delete - Use this link to delete the artifact version on this row.

Artifact Version Details

This page displays detailed information about an artifact version.

Links and actions at the top of the table




- **Edit** - Use this link to go to the Edit Artifact Version page.
- **Access Control** - Use this link to set privileges for this artifact version. *For more information, see the [Access Control](#) Help topic.*
- The "star" icon allows you to save this artifact version information to your Home page.

General Information section

This section lists current information about the artifact version. While most of the information is static, two items link to other pages:

Artifact - Click the artifact name to go to the Artifact Details page for the "owning" artifact.





Published By - Click this link to go to the Job Details page for the job that published this artifact version. If this artifact version was published by an end user outside of a job context, this field will display None.

Artifact Version Details – AM:Deploy:1.0.0-1   

General Information
Artifact: [AM:Deploy](#)
Group Id: AM
Artifact Key: Deploy
Version: 1.0.0-1
Published By: [job-21](#)
State: available
Description:

Files | Retrievers | Dependent Artifact Versions | Properties

Filename	Size
AM/Deploy/1.0.0-1	
commander	

1 of 1    

The "tabbed" sections

The next section of tabs allows you to select the type of information you want to see.

Files

This section shows the collection of files included in this artifact version. The left-pane displays the artifact version directory structure and the right-pane displays files and subdirectories for the currently selected folder in the left-pane. This structure is similar to Windows Explorer.

Files	Retrievers	Dependent Artifact Versions	Properties
<div> <div>AM/Deploy/1.0.0-1</div> <div>commander</div> </div> <div> <div>Filename</div> <div>Size</div> </div> <div> <div>commander</div> <div></div> </div> <div>1 of 1</div>			

Retrievers table

This table lists the 50 most recent jobs that retrieved this artifact version and includes links to the Job Details page for those jobs.

Files	Retrievers	Dependent Artifact Versions	Properties
<div>Job</div> <div>job_21_201108241328</div>			

Dependent Artifact Versions table

This section displays dependent artifact version queries registered with the artifact version during a publish operation. These queries are evaluated during retrieval and the most current artifact version that matches each query is returned along with the primary artifact version.

Files	Retrievers	Dependent Artifact Versions	Properties
<div>Dependent Artifact Versions</div> <div>AM:Lib</div>			

Properties table

This table contains custom properties for this artifact version.

- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Sheet**, and **Access Control** links to allow you to add more custom properties.

Files	Retrievers	Dependent Artifact Versions	Properties
<div> <a>Create Property <a>Create Nested Sheet <a>Access Control </div> <div> <div>Property Name</div> <div>Value</div> <div>Description</div> <div>Actions</div> </div> <div> <div>deployEuro</div> <div></div> <div></div> <div>Delete</div> </div> <div> <div>deployUS</div> <div></div> <div></div> <div>Delete</div> </div>			

Artifact Version - edit an existing artifact version

This page displays an existing artifact version you can modify.

Links and actions at the top of the table

- **Access Control** - Use this link to add or change permissions for this artifact version.
- The "star" icon allows you to save this page to your Home page for quick access.

To edit an artifact version

Field Name	Description / Action
Name	You can type-over the artifact version name in this field to change or modify the existing name.
Description	You can add a description or modify your existing description.
Publisher	This is the job that completed the publish operation. Click this link to go to the Job Details page for this job. If this artifact version was published outside of a job context (for example, from ectool), the value for this field is "None".
State	<p>An artifact version can be in one of three states: <code>available</code>, <code>unavailable</code>, and <code>publishing</code>.</p> <p>If the artifact version is:</p> <ul style="list-style-type: none">• in the Publishing state, the artifact version cannot be retrieved, but it will transition to the Available state when the publishing operation is complete. The web interface does not allow you to change the state if it is currently Publishing.• in the Available state, the artifact version can be retrieved. To confirm this state, the checkbox is "checked," and unchecking this box makes the artifact version unavailable.• in the Unavailable state, the artifact version cannot be retrieved. "Checking" the box changes the artifact version state to Available.

-

-

-

Repositories

This page displays all artifact repositories available to this Commander server.

Links and actions at the top of the table

- **Add Repository** - Use this link to go to the New Repository page to add another artifact repository.
- The "star" icon allows you to save this page to your Home page for quick access.

Column descriptions

Column Name	Description / Actions
Repository Name	<p>The name of the artifact repository.</p> <p>Select a repository name to go to the Edit Repository page if you need to modify that repository.</p> <p>Note: If you have multiple repositories and want to change the search order for retrieving artifacts, drag the icon (in the far left column) up or down to reposition an artifact to the order you need.</p>
Zone	<p>The name of the zone where this repository resides.</p>
URL	<p>The server URL is in the form <code>protocol://host:port/</code>. Typically, the repository server is configured to listen on port 8200 for <code>https</code> requests, so a typical URL looks like <code>https://host:8200/</code></p>
Enabled	<p>Select this checkbox to enable or disable this artifact repository. By disabling this repository, users will not be able to retrieve any artifacts stored in this repository.</p>
Description	<p>A plain text or HTML description previously supplied for this repository.</p>
Actions	<p>Delete - Use this link to delete the artifact repository on this row.</p>

Repository - create new or edit existing repository

To create a new repository

Fill-in the fields as follows:

Field Name	Description
Name	Supply any name of your choice for this repository.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
URL	The server URL is in the form <code>protocol://host:port/</code> . Typically, the repository server is configured to listen on port 8200 for <code>https</code> requests, so a typical URL looks like <code>https://host:8200/</code> .
Zone	Use the drop-down menu to select a zone for this repository. If no zone is selected, this repository will reside in the <i>default</i> zone.
Enabled	Select this checkbox to enable this repository. If this repository is disabled ("unchecked"), users will not be able to retrieve any artifacts stored in this repository.

Click **OK** after filling-in the fields and to return to the Repositories page to see your new repository listed in the first column.

To edit a repository

You can:

- Use the **Access Control** link at the top of the table to set permissions on this repository
- Change the repository name
- Add a description or modify the existing description
- Modify the URL
- Select a different zone.
- Enable or disable the repository

Click **OK** after making any modifications and to return to the Repositories page.

Database Configuration

If you chose not to use the ElectricCommander Build-in database, use this page to configure your MySQL, SQL Server, or Oracle database to communicate with Commander.

Note: The Commander "Built-in" database is not recommended for production use.

Fill in the fields as follows:

Field Name	Description
Database Type	Select your database type from the drop-down menu. If "Built-in" is selected, no other information is required.
Database Name	Supply your database name.
Host Name	Supply the host name for your database server.
Port	Use the default port or supply a new port number.
Database Credentials	User Name - Accept the default "commander" user name or supply the appropriate user name for your database. Password - Fill-in the database password.

IMPORTANT: When you set the SQL server as the database, you can only use a non-named database.

Click **Save and Restart Server** after supplying information in all fields.

Note: You may need to consult with your Database Administrator if you do not have all of the information required on this web page.

For more information, consult the *ElectricCommander Installation Guide*.

Defect Tracking Configurations

This page displays Defect Tracking configurations known to the ElectricCommander server—Commander integrates, using plugins, with numerous defect tracking systems.

[Link at the top of the table](#)

Use the **Create Configuration** link to create a new Defect Tracking configuration.

Column descriptions

Column Name	Description
Configuration Name	The name you provided to the configuration you created.
Description	This column describes the Defect Tracking configuration type.
Plugin	The name of the plugin where the Defect Tracking integration resides.
Actions	Edit - Click this link to modify an existing configuration. Delete - Click this link to remove the configuration.

Defect Tracking - create new or edit existing configuration

Use this page to define a new defect tracking configuration or modify an existing defect tracking configuration. A configuration is a collection of properties that define how Commander communicates with a particular defect tracking system.

After creating a defect tracking configuration, your entry will appear in the table on the Defect Tracking Configurations web page— to see this web page, select the Administration > Defect Tracking tabs.

To create a defect tracking configuration

From the Defect Tracking Type drop-down menu, select your defect tracking system—Commander integrates with numerous defect tracking systems. Each integration was created using plugin technology.

Click the **Submit** button and a set of fields is displayed to fill-in information to create your configuration.

Fill-in the fields as follows:

Field Name	Description
Configuration Name	Supply a unique name for this defect tracking configuration—any name you choose.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Login As	<p>User Name - This is the name Commander needs to use to communicate with your defect tracking system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.</p> <p>Password - This is the password for the specified User Name.</p> <p>Retype Password - Type the password again.</p> <p>URL - This is the URL to use to connect to a JIRA server. For example, you might use something similar to: <code>http://10.10.10.10:8080</code> or <code>http://yourJIRAserver</code></p>

Click **Submit** to save your information.

Using the defect tracking integration

Go to Projects > select a project > select a procedure.

- To create a New Step for defect tracking, select the Plugin link.
- In the Choose Step panel, select Defect Tracking from the left pane, then select the defect tracking system you configured.
- The right-pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the defect tracking integration you configured and the step you chose.
- Supply the remaining information to create your defect tracking step.

See the [Defect Tracking](#) help topic for more information.

To edit an existing defect tracking configuration

Modify any of your defect tracking configuration information by "typing-over" any previously entered information.

Click **Submit** to save your modified defect tracking information.

Defect Tracking Reports

This is a report of URLs to JIRA defects associated with the ElectricCommander job.

The report was compiled by searching job properties for JIRA defect IDs then querying the configured JIRA server for the current defect state.

For more information about Defect Tracking, see the main [Defect Tracking](#) help topic.

If you would like to create a defect tracking configuration, go to Administration > Defect Tracking and click the **Create Configuration** link to see the New Defect Tracking Configuration web page. Access the Help link on that page for more information.

Email Notifier - create new or edit existing email notifier

Use this page to set up email notifiers.

Note: You must have already set up an Email Configuration so ElectricCommander knows which email server to use when sending any notifications. If you have not configured an email server to communicate with ElectricCommander, click the Administration > Email Configuration tabs to do this configuration now.

For jobs and job steps - two types of email notifiers:

- On Start Notifier - sends an email when a job or job step starts.
- On Completion Notifier - sends an email when a job or job step completes.

For workflow states - three types of email notifiers:

- On Enter Notifier - sends an email when the state becomes the workflow's active state.
- On Start Notifier - sends an email after the state's subjob or subworkflow starts. If no subjob or subworkflow is defined for that state, these notifiers will not be sent.
- On Completion Notifier - sends an email after the state's subjob or subworkflow completes. If no subjob or subworkflow is defined for that state, these notifiers will not be sent.

Using email notifiers

- Attaching an Email Notifier to a procedure results in a corresponding Email Notifier associated with the job that is created when that procedure is executed.
- Attaching an Email Notifier to a procedure step results in a corresponding Email Notifier associated with the job step created when the procedure referencing the procedure step is executed.

Note: An Email Notifier attached to a procedure triggers email notifications only when that procedure is executed directly. No email notifications are sent when that procedure is invoked from a subprocedure step.

- Attaching an Email Notifier to a state definition results in a corresponding Email Notifier associated with the state that is created when that workflow is executed.

To create a new email notifier

Fill-in the fields as follows:

Field Name	Description
Name	This name can be an arbitrary text string.

Field Name	Description
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty or non-zero, the message is sent. If the result is "0", the message is not sent

Field Name	Description
Formatting Template	<p>Use the pull-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.</p> <p>To customize your template, edit the auto-supplied text in the Formatting Template text box or you can add a completely new script for your purpose. Any edits made in this text box will not be saved to the global template.</p> <p>Note: See the "Email notifier templates" section at the end of this help topic for a list of available templates and instructions for making global modifications to these templates.</p> <p>To create a custom template, the basic structure is:</p> <ul style="list-style-type: none"> • zero or more email header lines • blank line • message body <p>The template undergoes property expansion in a <code>job</code>, <code>jobStep</code>, or <code>state</code> context. In addition to normal property paths, an additional event object is accessible via <code>/myEvent/</code>.</p> <p>The following example is a sample formatting template for a job:</p> <pre>Subject: Job '\${jobName}' from procedure '\${procedureName}' \${/myEvent/type} - Commander notification cc: owner@example.org bcc: admin@example.org Commander Notification - from email notifier '\${/myEvent/notifier}' Job '\${jobName}' \${/myEvent/type} at: \${/myEvent/time} Project: \${projectName} Procedure: \${procedureName} Started: \${start} Completed: \${finish} Directory Name: \${directoryName} Launched by User: \${launchedByUser} Outcome: \${outcome} Error Code: \${errorCode} Error Message: \${errorMessage}</pre> <p>This example demonstrates how you can send this notification using CC or</p>

Field Name	Description
	BCC fields in addition to specified Destinations. Because headers are interpreted by SMTP, normal email rules apply.
Email Configuration	Click inside this field or start typing to bring up a list of possible email configuration names. An email notifier that does not specify an email configuration will use the configuration named 'default' if it exists.
Destinations	A space-separated list of valid email addresses, email aliases, or ElectricCommander user or group names, or a property reference that expands into such a list, or you can supply an LDAP DL name (group name).

Click **OK** to save your email notifier configuration.

To edit an existing email notifier

Modify or add information to any of the fields and click **OK** to save your changes.

Email notifier templates

The table below lists global email notifier templates for your use. Each template name is a link to an example of the template output and the script.

Global templates are stored as property sheets under the `"/server/ec_notifierTemplates"` property sheet. You can modify template properties or add new global templates if you have modify privileges on the server property sheet. An email notifier will be updated if changes are made to its global template.

Template Name	Format	Usage	Description
Job Summary text notification	Plain text	Procedure	Formatting template for Procedure starting/completion notifications using the long form of property names (for example, <code>\$/myJob/jobName</code>)
Step summary text notification	Plain text	Procedure step	Formatting template for Procedure Step starting/completion notifications using the long form of property names (for example, <code>\$/myJobStep/jobStepId</code>)
State summary text notification	Plain text	State	Formatting template for State notifications using the long form of property names (for example, <code>\$/myWorkflow/workflowName</code>)
Job summary HTML notification	HTML	Procedure	HTML formatting template for Procedure starting/completion notifications
Step summary HTML notification	HTML	Procedure step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and the Job Step

Template Name	Forma- t	Usage	Description
All steps HTML notification	HTML	Procedur e step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and many Job Steps
Approval request HTML notification	HTML	State	HTML formatting template for State notifications displaying information about the workflow and all of its states
Workflow summary HTML notification	HTML	State	HTML formatting template for State notifications providing the recipient with links to view or take a manual transition

Email Configurations

This page displays all previously configured email configurations.

- Click on a Configuration Name to go to the Edit Email Configuration web page to modify an existing email configuration.
- Click on the Add Configuration link to create new email configuration.
- Action column - This column contains two links to Copy or Remove an Email Configuration.

Notice this page also lists the email server name and who the email notification was from.

Email Configuration - create new or edit existing email configuration

Use this web page to specify an email server to ElectricCommander. If you do not create an Email Configuration, you will not be to send Email Notifications to individuals or groups.

If you have multiple users or groups in remote locations that use a different mail server, create additional Email Configurations to accommodate those locations if they need to receive Commander notifications.

To create a new email configuration

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique name to identify the Email Configuration.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Mail Protocol	Either SSMTP or SMTP
Mail Host	The name of the mail server.
Mail Port	The port number for the mail server, but may not need to be specified. 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.
Mail From	An individual name or a generic name like "Commander" or "build."
Mail Username	An individual name or a generic name like "Commander."
Mail Password	The password for the mail user name account.

Click **Save** after filling-in all fields.

Click the **Test** button to run a test to make sure your email configuration works. Supply an email address in the pop-up box and click **Send**.

To edit an existing email configuration

Use the **Access Control** link at the top of the page to add or change existing permissions.

You can modify any of the information already supplied or add new information.

Click **OK** after completing your edits.

Email Notifier - create new or edit existing email notifier

Use this page to set up email notifiers.

Note: You must have already set up an Email Configuration so ElectricCommander knows which email server to use when sending any notifications. If you have not configured an email server to communicate with ElectricCommander, click the Administration > Email Configuration tabs to do this configuration now.

For jobs and job steps - two types of email notifiers:

- On Start Notifier - sends an email when a job or job step starts.
- On Completion Notifier - sends an email when a job or job step completes.

For workflow states - three types of email notifiers:

- On Enter Notifier - sends an email when the state becomes the workflow's active state.
- On Start Notifier - sends an email after the state's subjob or subworkflow starts. If no subjob or subworkflow is defined for that state, these notifiers will not be sent.
- On Completion Notifier - sends an email after the state's subjob or subworkflow completes. If no subjob or subworkflow is defined for that state, these notifiers will not be sent.

Using email notifiers

- Attaching an Email Notifier to a procedure results in a corresponding Email Notifier associated with the job that is created when that procedure is executed.
- Attaching an Email Notifier to a procedure step results in a corresponding Email Notifier associated with the job step created when the procedure referencing the procedure step is executed.

Note: An Email Notifier attached to a procedure triggers email notifications only when that procedure is executed directly. No email notifications are sent when that procedure is invoked from a subprocedure step.

- Attaching an Email Notifier to a state definition results in a corresponding Email Notifier associated with the state that is created when that workflow is executed.

To create a new email notifier

Fill-in the fields as follows:

Field Name	Description
Name	This name can be an arbitrary text string.

Field Name	Description
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty or non-zero, the message is sent. If the result is "0", the message is not sent

Field Name	Description
Formatting Template	<p>Use the pull-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.</p> <p>To customize your template, edit the auto-supplied text in the Formatting Template text box or you can add a completely new script for your purpose. Any edits made in this text box will not be saved to the global template.</p> <p>Note: See the "Email notifier templates" section at the end of this help topic for a list of available templates and instructions for making global modifications to these templates.</p> <p>To create a custom template, the basic structure is:</p> <ul style="list-style-type: none"> • zero or more email header lines • blank line • message body <p>The template undergoes property expansion in a <code>job</code>, <code>jobStep</code>, or <code>state</code> context. In addition to normal property paths, an additional event object is accessible via <code>/myEvent/</code>.</p> <p>The following example is a sample formatting template for a job:</p> <pre>Subject: Job '\${jobName}' from procedure '\${procedureName}' \${/myEvent/type} - Commander notification cc: owner@example.org bcc: admin@example.org Commander Notification - from email notifier '\${/myEvent/notifier}' Job '\${jobName}' \${/myEvent/type} at: \${/myEvent/time} Project: \${projectName} Procedure: \${procedureName} Started: \${start} Completed: \${finish} Directory Name: \${directoryName} Launched by User: \${launchedByUser} Outcome: \${outcome} Error Code: \${errorCode} Error Message: \${errorMessage}</pre> <p>This example demonstrates how you can send this notification using CC or</p>

Field Name	Description
	BCC fields in addition to specified Destinations. Because headers are interpreted by SMTP, normal email rules apply.
Email Configuration	Click inside this field or start typing to bring up a list of possible email configuration names. An email notifier that does not specify an email configuration will use the configuration named 'default' if it exists.
Destinations	A space-separated list of valid email addresses, email aliases, or ElectricCommander user or group names, or a property reference that expands into such a list, or you can supply an LDAP DL name (group name).

Click **OK** to save your email notifier configuration.

To edit an existing email notifier

Modify or add information to any of the fields and click **OK** to save your changes.

Email notifier templates

The table below lists global email notifier templates for your use. Each template name is a link to an example of the template output and the script.

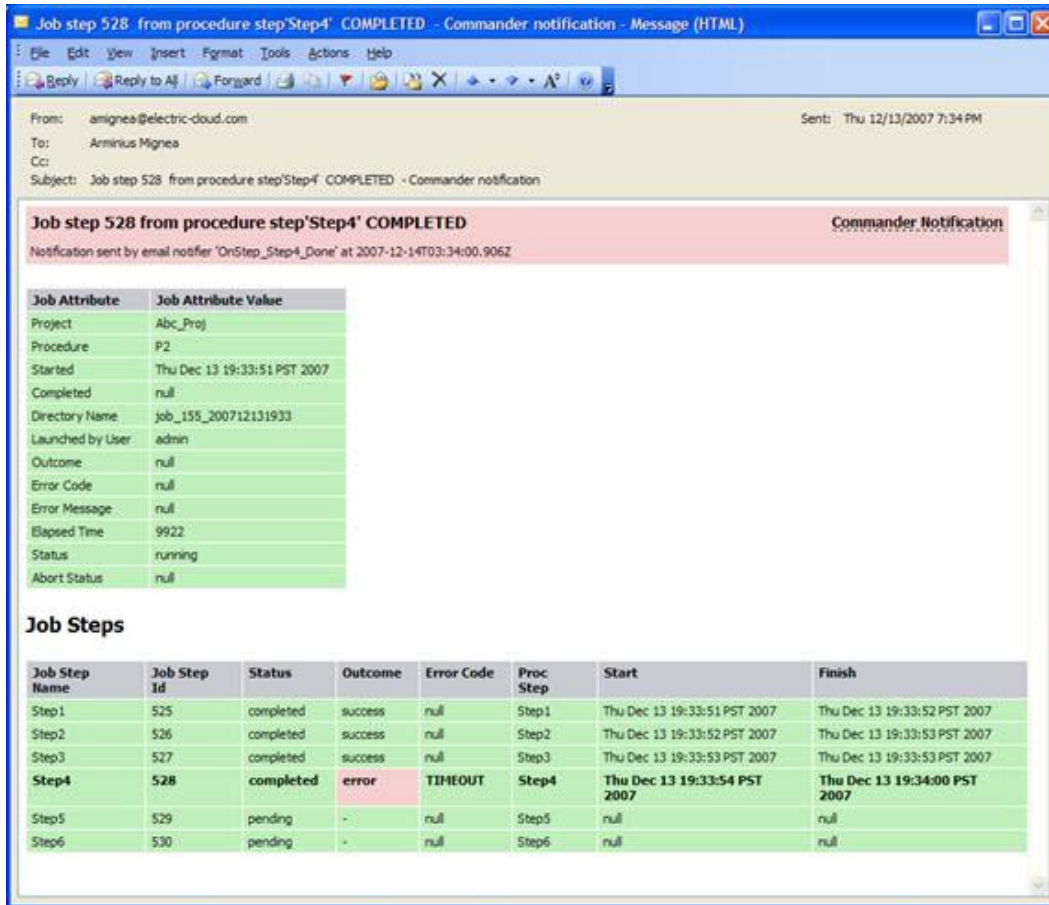
Global templates are stored as property sheets under the `"/server/ec_notifierTemplates"` property sheet. You can modify template properties or add new global templates if you have modify privileges on the server property sheet. An email notifier will be updated if changes are made to its global template.

Template Name	Forma- t	Usage	Description
Job Summary text notification	Plain text	Procedur e	Formatting template for Procedure starting/completion notifications using the long form of property names (for example, <code>\$/myJob/jobName</code>)
Step summary text notification	Plain text	Procedur e step	Formatting template for Procedure Step starting/completion notifications using the long form of property names (for example, <code>\$/myJobStep/jobStepId</code>)
State summary text notification	Plain text	State	Formatting template for State notifications using the long form of property names (for example, <code>\$/myWorkflow/workflowName</code>)
Job summary HTML notification	HTML	Procedur e	HTML formatting template for Procedure starting/completion notifications
Step summary HTML notification	HTML	Procedur e step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and the Job Step

Template Name	Format	Usage	Description
All steps HTML notification	HTML	Procedure step	HTML formatting template for Procedure Step starting/completion notifications displaying information about the Job and many Job Steps
Approval request HTML notification	HTML	State	HTML formatting template for State notifications displaying information about the workflow and all of its states
Workflow summary HTML notification	HTML	State	HTML formatting template for State notifications providing the recipient with links to view or take a manual transition

Email Notifier Template - Html_JobStepTempl_AllSteps.txt

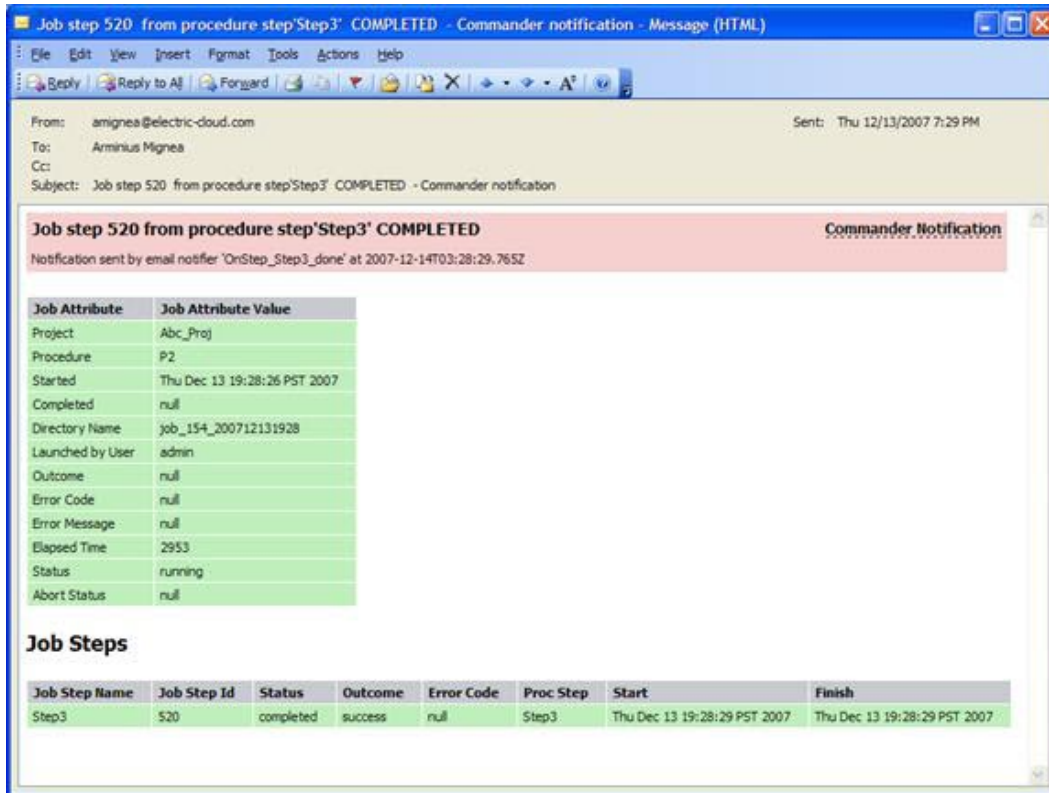
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - Html_JobStepTempl_SingleStep.txt

If you use this template, the following screen example is similar to how your email notifier will look.

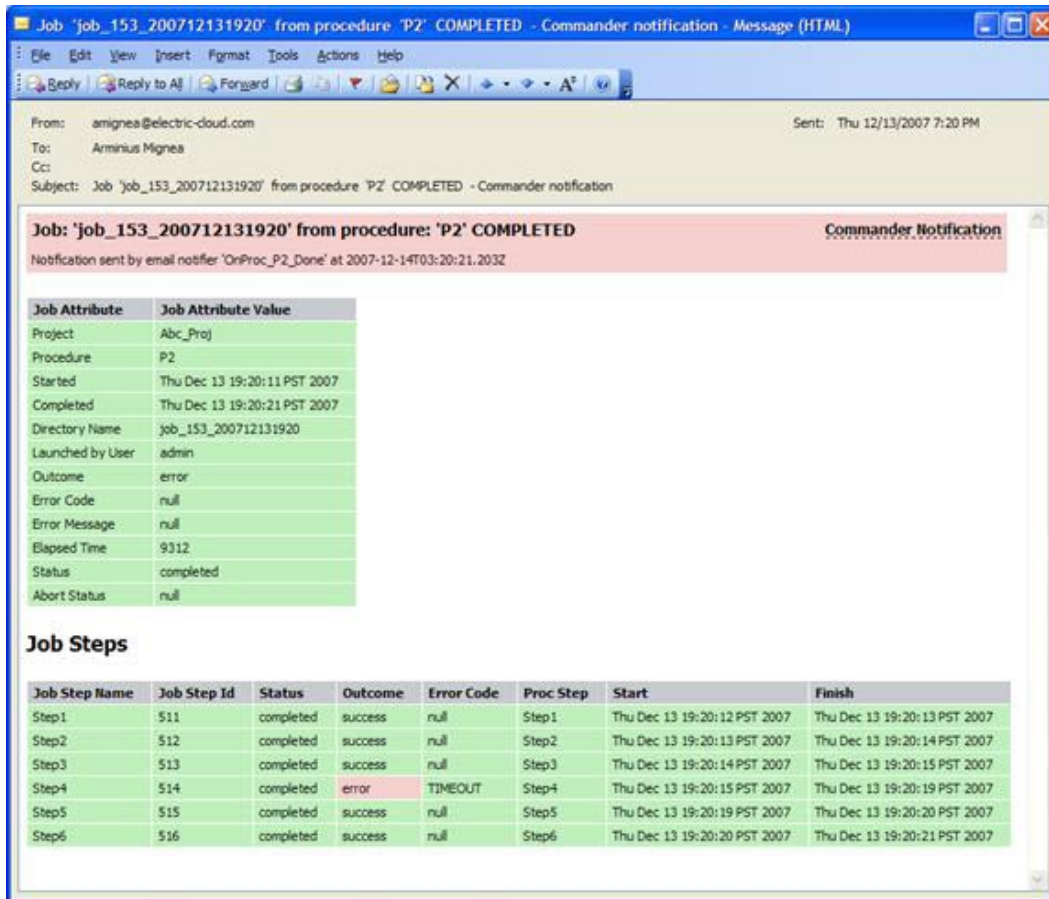


Note: In the following script, myJob.steps and myJob.jobSteps return an array of step names.

1. Go to the corresponding template available in <installDir>\src\samples\notifier
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - Html_JobTempl.txt

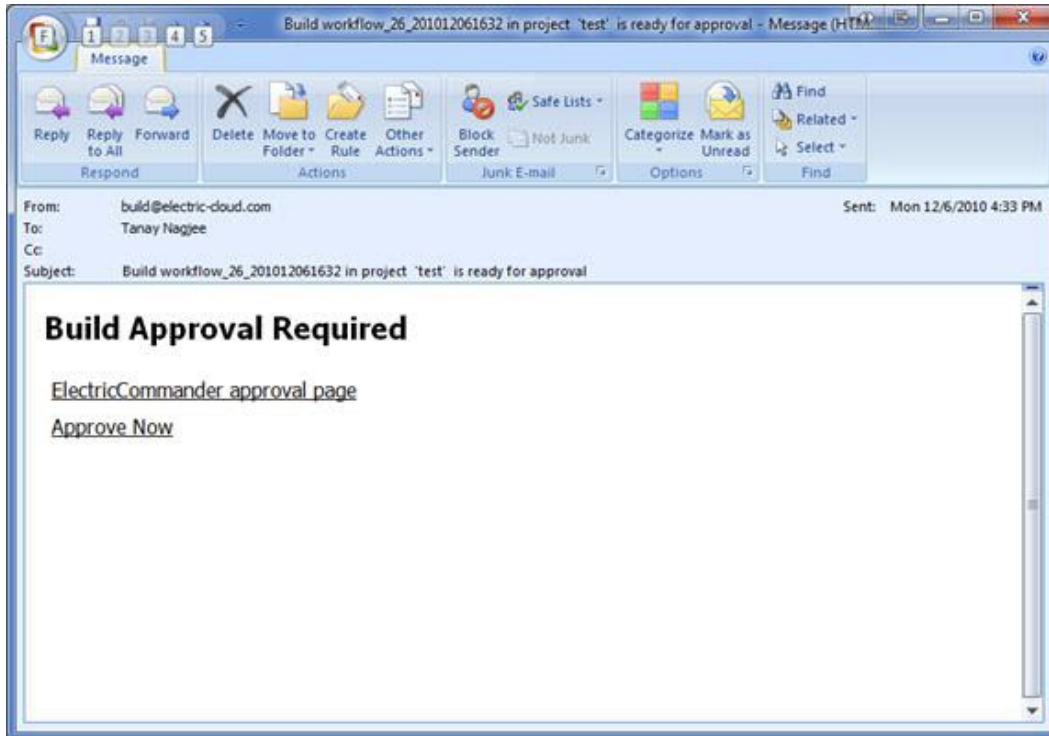
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - Html_StateTemplate_ApproveWorkflow.txt

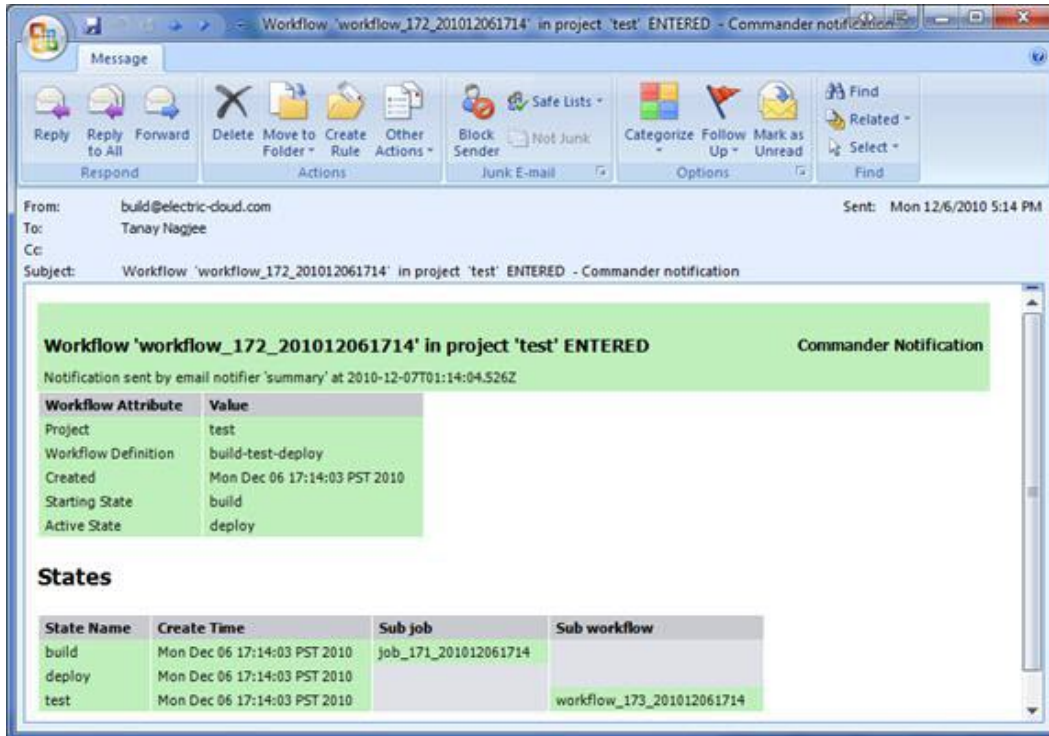
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - Html_StateTemplate_FullWorkflow.txt

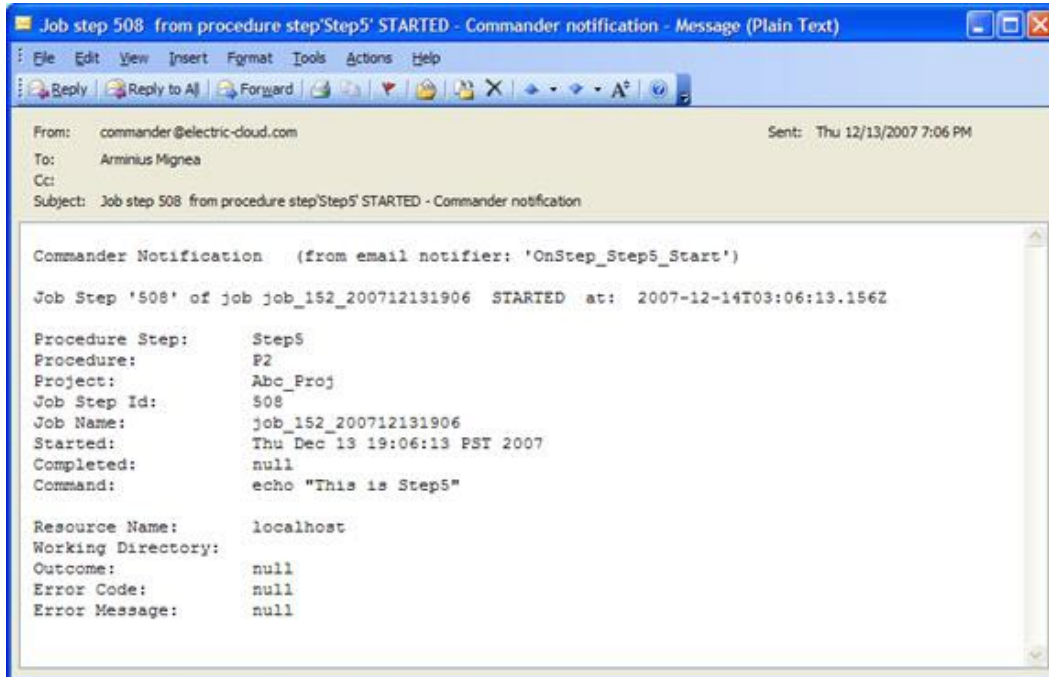
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - JobStepTempl_FullProps.txt

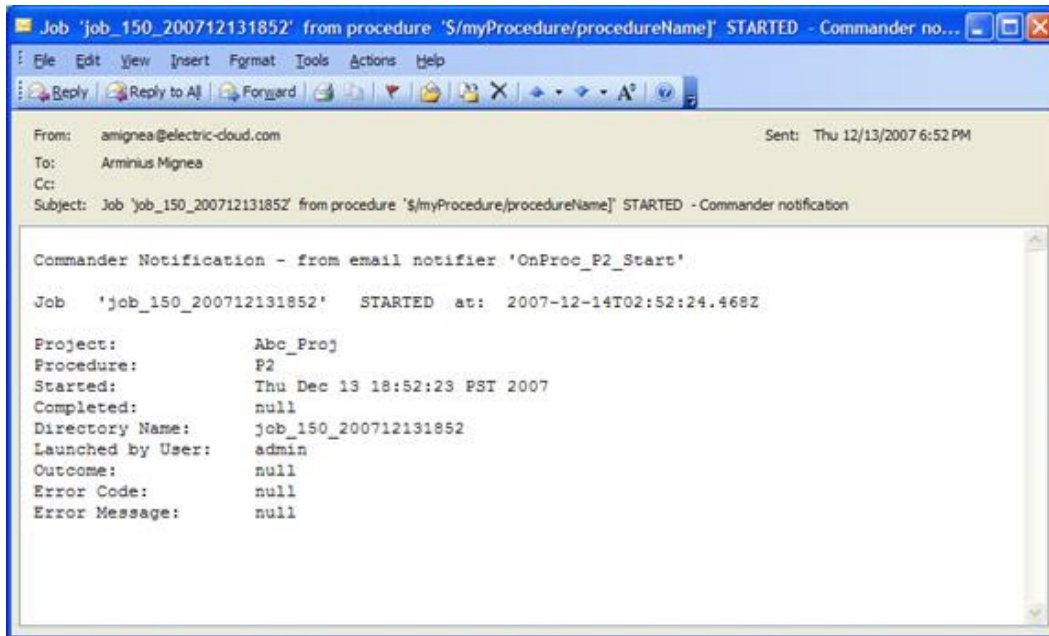
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - JobTempl_FullProps.txt

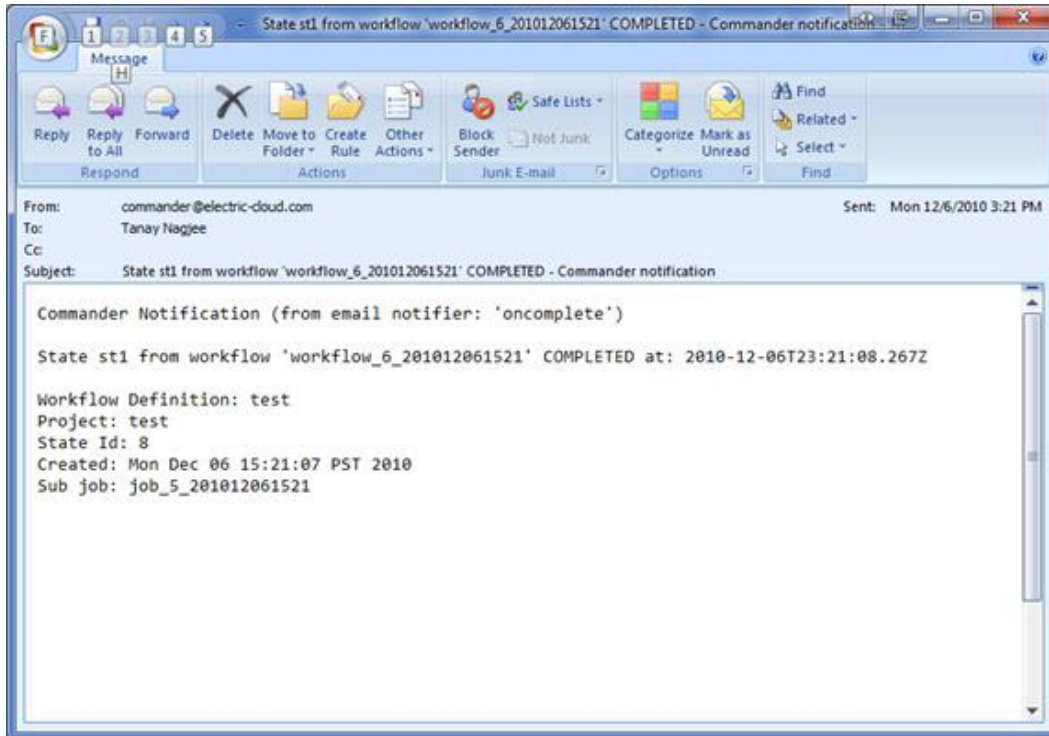
If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Email Notifier Template - StateTemplate_FullPropertyPaths.txt

If you use this template, the following screen example is similar to how your email notifier will look.



1. Go to the corresponding template available in `<installDir>\src\samples\notifier`
2. Copy the script into a text editor to edit it for your purpose
3. "Copy and paste" your version into the Formatting Template field.

Event Log

This page displays a log of events generated anywhere in the system, including jobs and workflows.

- 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.

Note: When running a job in a cluster, the name of the server managing the job is recorded in the job-log. If the management of the job moves to a different node (if the current node crashes for example) the change is logged into the job log as well.

- To see only events for a specific object, select the Search tab to go to the Define Search page.

For example:

You might want to select the Object Type, "Log Entry", then click the **Add Intrinsic Filter** link. Select the down-arrow where you see "Container" auto-populated and select "Container Type". Use the "equals" operator, then select the next down-arrow to choose an object. Click **OK** to start the search.

Note:

From the Administration tab, the default view for this 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.

Column descriptions

Time - This is the time the event was logged.

Severity - Severity can be either INFO, WARN, or ERROR. Use the pull-down menu at the top of the table to change between these views.

User - The user associated with the event—the user of the session at the time the event was generated. Clicking on a project link in this column takes you to the Project Details page for that project.

Container - Typically, this is the type and name of the workflow or job with a corresponding ID. Clicking on a link in this column takes you to the Job Details page for that job or workflow.

Subject - The object associated with the message. Clicking on a link in this column takes you to the source for this object.

Message - The message text will either be informational or display the warning or error message. The message may contain important information about a resource or workspace issue.

Configuring the Event Log

Because log entries can accumulate in large numbers, you can configure the Event Log for auto-deletion. By default, Event Log retention is 30 days. The Commander server automatically marks old log entries for deletion and the background deleter cleans out old log entries along with its other tasks.

To change the default settings, go to Administration > Server > Settings.

Modify the following two fields:

Event log retain time - Default is 30 days (in "minute" units).

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Maximum background delete delay - Default is 3600 seconds (or one hour).

Click **OK** to save your changes.

Jobs

This page displays all jobs in the ElectricCommander system, both running and completed jobs. You can:

- Sort the Job, Status, Elapsed Time, and Start Time columns by clicking on the column name.
- Define a search to see only the jobs you choose.

Links at the top of the table

- **New Search** - click this link to go to the Define Search page to select a specific set or range of jobs to view.
- **Edit Search** - (available after a "search" is defined) click this link to redefine your search.
- **Refresh Results** - click this link to see updated status for running jobs.

Column descriptions

Job - click on Job to sort the column alphabetically or click on a job name to go to that job's Job Details page.

Status - click on Status to sort this column by Running, Success, Warning, Error, or Aborted.

Priority - displays the job's priority set by a "run procedure" command or by the job's schedule.

Procedure - in this column you can click on a project name to go to the Project Details page or click on the procedure name to go to the Procedure Details page for that procedure.

Launched By - click on a name in this column to go to the page for the schedule (Edit Schedule page) or the user that ran the job.

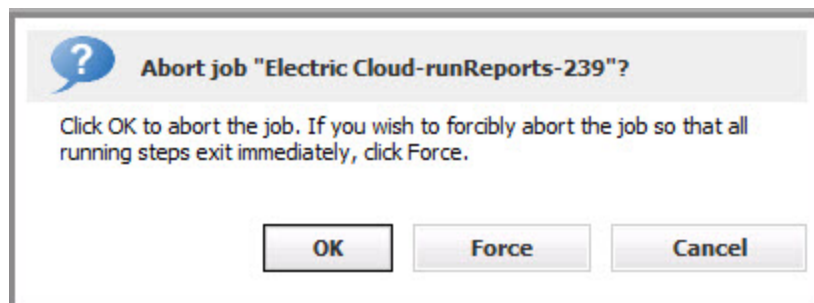
Elapsed Time - click Elapsed Time to sort the time values from longest to shortest time, or the reverse.

Start Time - click Start Time to sort this column from the start time of the first job to the start time of the most recent job, or the reverse.

Actions - use this column to Abort a running job or Delete a completed job.

- Aborting a job requires the execute privilege on the job—not just the modify privilege.

When you click Abort, the following dialog box appears:



Select either **OK** or **Force** to abort the job, or **Cancel** if you change your mind.

- Deleting a job on this page causes removal of job information from the Commander database, but information in the job's on-disk workspace area is not affected. You must delete workspace information manually.

Tips

If you would like to select a subset of the jobs, use the **Search** link at the top of the page.

From your Home page, you can add a specific "Search" for quick access on a regular basis.

Also notice: If you use RSS, an active RSS icon is provided in Windows Explorer on this page for your convenience. If you use Firefox, click **Bookmarks** > Subscribe to this page to display the feed. You can then add the feed URL to a viewer of your choice.

Jobs

This page displays all jobs in the ElectricCommander system, both running and completed jobs. You can:

- Sort the Job, Status, Elapsed Time, and Start Time columns by clicking on the column name.
- Define a search to see only the jobs you choose.

Links at the top of the table

- **New Search** - click this link to go to the Define Search page to select a specific set or range of jobs to view.
- **Edit Search** - (available after a "search" is defined) click this link to redefine your search.
- **Refresh Results** - click this link to see updated status for running jobs.

Column descriptions

Job - click on Job to sort the column alphabetically or click on a job name to go to that job's Job Details page.

Status - click on Status to sort this column by Running, Success, Warning, Error, or Aborted.

Priority - displays the job's priority set by a "run procedure" command or by the job's schedule.

Procedure - in this column you can click on a project name to go to the Project Details page or click on the procedure name to go to the Procedure Details page for that procedure.

Launched By - click on a name in this column to go to the page for the schedule (Edit Schedule page) or the user that ran the job.

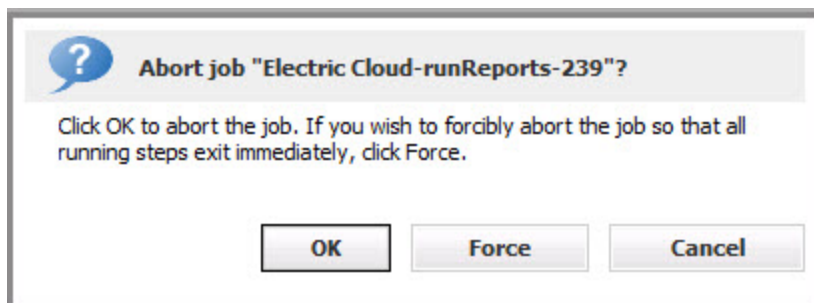
Elapsed Time - click Elapsed Time to sort the time values from longest to shortest time, or the reverse.

Start Time - click Start Time to sort this column from the start time of the first job to the start time of the most recent job, or the reverse.

Actions - use this column to Abort a running job or Delete a completed job.

- Aborting a job requires the execute privilege on the job—not just the modify privilege.

When you click Abort, the following dialog box appears:



Select either **OK** or **Force** to abort the job, or **Cancel** if you change your mind.

- Deleting a job on this page causes removal of job information from the Commander database, but information in the job's on-disk workspace area is not affected. You must delete workspace information manually.

Tips

If you would like to select a subset of the jobs, use the **Search** link at the top of the page.

From your Home page, you can add a specific "Search" for quick access on a regular basis.

Also notice: If you use RSS, an active RSS icon is provided in Windows Explorer on this page for your convenience. If you use Firefox, click **Bookmarks** > Subscribe to this page to display the feed. You can then add the feed URL to a viewer of your choice.

Job Details

This page displays detailed information about a job. If the job is currently running, the page updates automatically as the job progresses. After the job is complete, the page will not continue to refresh. You can disable the automatic page refresh by clicking the **Stop Refresh** link at the top of the page.

Links and actions at the top of the table

- **Abort** - Aborts a running job.
- **Run Again** - Use this link to run the job again.
When you select the **Run Again** button, you are requesting to re-run the job with the same parameter values as the original invocation.
If you change the parameter values, you can select "**Run...**" from the drop-down menu (small down-arrow) next to the Run Again button.
- **Delete** - Deletes this job.
- **Save Configuration** - Saves this configuration to the Quick View section on the Home page.
- **Access Control** - Use this link to set privileges for this job. For more information, see the [Access Control](#) Help topic.
- **Stop Refresh** - This link is available only if a job is currently running. Click this link if you do not want the page to refresh automatically while a job is running.
- **View Log** - This link takes you to the Job Log page. If your job "completed with errors", this log contains error messages.
- The "star" icon allows you to save this job information to your Home page.
- The "curved arrow" icon returns you to the Jobs page.
- **Pagination** - Use the "previous" and "next" arrow icons to view the previous or next job. The numbers between the arrow icons display the number of jobs you can view and the first number indicates which job [in the list] you are viewing.

Summary section (at the top of the page)

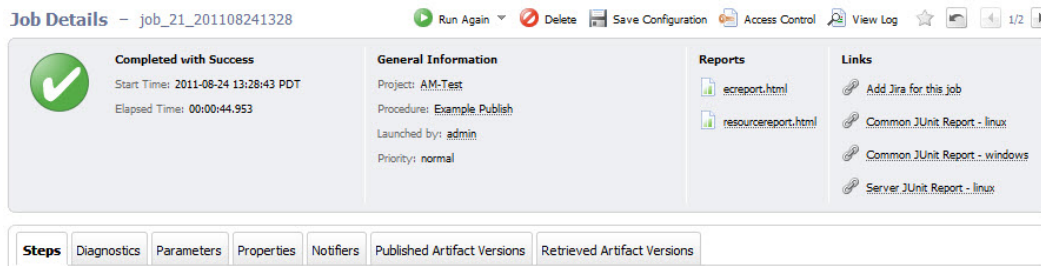
Along with the job's start and elapsed times, the summary section provides links to get you to the job's Project or Procedure Details pages or to the Edit Schedule page.

This section also displays:

- A job's priority and a "Calling State" category - If the job was called by a workflow, you will see two links, one to the Workflow Details page and one to the State Details page.
- "Wait times" - You will see *wait times* specified if your running job was restricted by resource, workspace, a precondition, or license availability.
Note: Total job wait time is more accurately thought of as *accumulated* wait time because it is the sum of all step wait times, including wait times for steps running in parallel.
- Two other useful sections: Reports and Links. If the job created any custom reports, they appear in the summary area to the right of **General Information**.

Reports

To create a special-purpose HTML report for a job: In the top-level of any of the job's workspaces, add a file that contains the word "report" or "Report" and ends in the .html file extension. The link is automatically created when the Job Details page is displayed. The job's workspace must be accessible to the web server. The link will be in the Reports section. The screen capture below displays an example of the HTML Reports section.



Click on a report file name to display the report.

Links

Notice the additional Links section in the screen example above. To add a link to this section:

- **Create a link to any file:**

To add a link, run a command in any job's job step using this format (this works for both local [disconnected] and non-local workspaces):

```
ectool setProperty "/myJobStep/report-
urls/<
myReportName>"/commander/jobSteps/<jobStepId>/<artifactDirectoryName>/<file-
path>
```

Example command:

```
ectool setProperty "/myJobStep/report-urls/Test Report"/commander/jobSteps/
$[/myJobStep/jobStepId]/testreport/index.html
```

Note: If you are using this format (from Commander versions prior to v4.2):

```
ectool setProperty "/myJob/report-urls/<myReportName>"/commander/jobs/<jobId>/<w
orkspaceName>/<artifactDirectoryName>/<file-path>
```

Example command:

```
ectool setProperty "/myJob/report-urls/Test Report"/commander/jobs/$[/myJob/jobI
d]/
$[/myJobStep/workspaceName]/testreport/index.html
```

it will continue to work for non-local workspaces only. It is recommended, however, to change the command to use the newer format.

- **Create a link to any directory:**

To add a link, run the following style command in any job's job step:

```
ectool setProperty "/myJobStep/report-urls/Main Job Workspace"
"file:///WinStor2/scratch/chron55build/${/myJob/jobName}"
```

Note: Links to a directory automatically work in Internet Explorer, but if using Firefox, local links are disabled unless the default security policy is modified or an extension is used. Refer to http://kb.mozillazine.org/Links_to_local_pages_don%27t_work for more details.

The "tabbed" sections

The next section of tabs allows you to select the type of information you want to see.

Steps table

The Job Steps table displays all steps for this job.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Retrieved Artifact Versions
View: All Expand All Collapse All						
Step Name	Log	Status	Elapsed Time	Resource	Actions	
Retrieve-Lib		Success	00:00:01.135		Edit	
Retrieve Artifact Version		Success	00:00:01.103	local	Edit	
Build		Success	00:00:05.255	local	Edit	
Publish		Success	00:00:20.576		Edit	
Publish Artifact Version		Success	00:00:20.547	local	Edit	
Reports		Success	00:00:02.639	local	Edit	
Links		Success	00:00:02.494	local	Edit	

- Mouse-over the **View: All** link [below the tabs] to see a filter list. You can choose a filter to apply to the Job Step table.
- The **Expand All | Collapse All** links are provided to see all subprocedures or to minimize the information presented.
- Click on a Step Name to go to the Job Step Details page for that step.
- Click the Log icon for any step to display the step's log file. This icon is not displayed if the `logFileName` property was removed from the step.
- Status column: This column provides the job's step status.
 - Aborted - The step or job was aborted.
 - Canceled - The step did not run because the job was aborted or a previous step was aborted.
 - Error - The command failed or `postp` detected an error in the output.
 - Running - The step is currently running or a step inside the subprocedure call is running.
 - Skipped - The step did not run because the run condition evaluated to "false".
 - Success - The step completed with no errors or warnings.
 - Waiting for Resource or Workspace - The step is waiting to run as soon as a resource or workspace is available.
 - Warning - `postp` detected a warning message.

Note: The Status field can be over-ridden by `postp` to include more specific information. `postp` information may be linked directly to diagnostic information—also available by selecting the Diagnostics tab.

Action column:

- Click the **Edit** link to edit the procedure step definition.
- Aborting a step requires the execute privilege on the job—not just the modify privilege.

When you click **Abort**, the following dialog box appears:



Click **OK** to abort the step. If the step does not complete in approximately two minutes, you can "forcibly" abort the step to force the step to a completed status, enabling the rest of the job to continue.

Click **Cancel** if you change your mind.

Diagnostics table

If the job generated warning or error messages, click this tab to see one or more diagnostics, each of which is a portion of a step's log file identified as "interesting" by the step's postprocessor. Typically, each diagnostic relates to an error or warning detected by the postprocessor. All diagnostics for a particular step appear together.

Each diagnostic contains a header line such as "Error #3 of 5" with an icon, followed by a line with additional identifying information and one or more lines extracted from the log file. The identifying line has a format similar to:

```
"systemTest-windows.log:16 (systemtest) "
```

- The first part of the line provides the name of the log file (`systemTest-Windows.log`) and the line number of the first line in the file that is part of this diagnostic (16).
- If you click on the log file name, Commander displays the full log file for you, with the portion in this diagnostic highlighted. If the postprocessor was able to provide additional information about the issue, it is displayed in parentheses after the log filename. The first value (`systemtest-windows`) gives the name of a particular module or package in which the problem occurred.
- The second value (`systemtest`) identifies a specific place where the problem occurred, such as the file name that did not compile or the name of a test that failed.

Steps **Diagnostics** Parameters Properties Notifiers Published Artifact Versions Retrieved Artifact Versions

View: All Messages 1-20 of 44

Step 'Unit tests - windows' - 26 Warnings

Warning #1 of 26 in step 'Unit tests - windows'
 unittest-win.log:134 (EC-CreateProcedureWizard)

compile.production:
 compile.tests:
 [copy] Copying 13 files to u:\plugins-martin-main-scm-systemtests.683-201003291444\out\EC-CreateProcedureWizard\transform\test
 [javac] Warning: com\electriccloud\commander\plugins\EC_CreateProcedureWizard\client\CreateProcedureWizardSubpanelTest.java modified i

Warning #2 of 26 in step 'Unit tests - windows'
 unittest-win.log:139 (EC-CreateProcedureWizard)

[javac] Warning: com\electriccloud\commander\plugins\EC_CreateProcedureWizard\client\CreateProcedureWizardTestBase.java modified i

Warning #3 of 26 in step 'Unit tests - windows'
 unittest-win.log:140 (EC-CreateProcedureWizard)

[javac] Warning: com\electriccloud\commander\plugins\EC_CreateProcedureWizard\client\FakeGlobalStringManager.java modified in the

Warning #4 of 26 in step 'Unit tests - windows'
 unittest-win.log:141 (EC-CreateProcedureWizard)

[javac] Warning: com\electriccloud\commander\plugins\EC_CreateProcedureWizard\client\FinishPanelTest.java modified in the future.

Parameters table

This section displays parameter values supplied to the top-level procedure when the job was invoked. This table does not allow you to edit the parameter's value.

These parameters are copied automatically to the top-level property sheet and you will see them in the properties table also.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Retrieved Artifact Versions
Name	Value					
branch	main					
buildResource	ec-lin					
generateReport	1					
preFlight	1					
preFlightTag	scm-systemtests					

Properties table

This table contains information computed and/or used by the job while it is running.

- All defined parameters are automatically copied to the top-level property sheet. Any custom properties defined for the state will be included in this table also.
- After the job completes, these properties are normally read-only.
- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- If you use square brackets in a property name, the property must be encapsulated in square brackets to avoid property sheet expansion. Note that this works for specific functions, such as creating a link on a job, but the drawback is that the property cannot be edited or deleted from the web interface. Example:

```
ectool setProperty "/myJob/report-urls[This property name has [some square bracket] text in it]" "http://website.com"
```

- This section also provides **Create Property**, **Create Nested Sheet**, and **Access Control** links to update or supply additional information for the next time this job is run.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Retrieved Artifact Versions
Create Property Create Nested Sheet Access Control						
Property Name	Value	Description	Actions			
<input type="checkbox"/> LabManager			Edit Delete			
<input type="checkbox"/> report-urls			Edit Delete			
RUNSCMTESTS_lin	0		Delete			
RUNSCMTESTS_win	1		Delete			
branch	main		Delete			
buildNumber	683		Delete			
buildResource	ec-lin		Delete			

Notifiers table

If you designated certain persons or groups to be notified of details for specific jobs, this section displays who receives an email notification, the type of notification, any condition, and a description [if supplied]. If a notifier is available, you can click on its name to view details or make modifications.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Retrieved Artifact Versions
Name	Type	Condition	Description	Actions		
Chronic3 Sentry failed	onCompletion	<code>\$/javascript if(getProperty("outcome") == 'error') true; else false;</code>	Notify me if chronic3 sentry fails	Delete		

Published Artifact Versions table

This table displays artifact versions published by steps in this job.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Retrieved Artifact Versions
Name	State	Description	Actions			
AMrDeploy:1.0.0-1	available		Edit Delete			

Column descriptions

- **Name** - Select this name to go to the Artifact Version Details page for this artifact version.
- **State** - Possible values for the state of the artifact version are: `available|publishing|unavailable`
- **Description** - A previously supplied text or HTML description for this artifact version.
- **Actions column:**
 - **Edit** - Select this link to go to the Edit Artifact Versions page.
 - **Delete** - Select this link to delete the artifact version. You may be prohibited from deleting this artifact version if you do not have sufficient access control permissions.

Retrieved Artifact Versions table

This table displays artifact versions retrieved by steps in this job.

Steps

Diagnostics

Parameters

Properties

Notifiers

Published Artifact Versions

Retrieved Artifact Versions

Name	State	Description	Actions
AM:Deploy:1.0.0-1	available		Edit Delete
AM:Lib:1.0.0-1	available		Edit Delete

Column descriptions

- **Name** - Select this name to go to the Artifact Version Details page for this artifact version.
- **State** - Possible values for the state of the artifact version are: `available|publishing|unavailable`
- **Description** - A previously supplied text or HTML description for this artifact version.
- **Actions column:**
 - **Edit** - Select this link to go to the Edit Artifact Versions page.
 - **Delete** - Select this link to delete the artifact version. You may be prohibited from deleting this artifact version if you do not have sufficient access control permissions.

Job Step Details

This page displays detailed information about a job step.

Links and actions at the top of the table

- **Access Control** - Use this link to set privileges for this job step. For more information, see the Access Control Help topic.
- "star" icon - click this icon to add this page to your Home for quick one-click access in the future.

Summary section (at the top of the page)

Notice [in the screen example below] that the name of the job step you are viewing is adjacent to the page title, Job Step Details.

The Summary section provides:

- The job's Start, Wait (waiting for a resource), and Elapsed times
- You will see wait times specified if your job steps were restricted by resource, workspace, a precondition, or license availability.
For example, you will see the license wait time if a job step could not run because your license-allowed maximum concurrent step limit was met or exceeded, meaning no steps will run until the number of concurrent steps is reduced.
- A General Information section with links to get you to the job step's Job Details, Project Details, Procedure Details, or Edit Step pages.
- If the job created any custom reports, they appear in the summary area to the right of General Information. Reports on this page are for command steps only.

Reports

To create a special-purpose HTML report for a job: In the top-level of any of the job's workspaces, add a file that contains the word "report" or "Report" and ends in the .html file extension. The link is automatically created when the Job Details page is displayed. The job's workspace must be accessible to the web server. The link will be in the Reports section, within the Summary section. (The screen capture below does not illustrate the Reports section, which would be to the right of the General Information section.)

Job Step Details - Publish-Lib  Access Control 

 Completed with Success Start Time: 2011-08-25 10:52:38 PDT Wait Time: 00:00:00.401 Elapsed Time: 00:00:16.139	General Information Job: job_1_201108251052 Project: AM-Test Procedure: Example Publish lib Step: Publish-Lib
---	--

Links

This page can include a Links section also, which will appear on the far right-side of the Summary section.

To add a link to this section:

- **Create a link to any file:**

To add a link, run a command in any job's job step using this format (this works for both local [disconnected] and non-local workspaces):

```
ectool setProperty "/myJobStep/report-  
urls/<  
myReportName>"/commander/jobSteps/<jobStepId>/<artifactDirectoryName>/<file-  
path>
```

Example command:

```
ectool setProperty "/myJobStep/report-urls/Test Report"/commander/jobSteps/  
$[/myJobStep/jobStepId]/testreport/index.html
```

Note: If you are using this format (from Commander versions prior to v4.2):

```
ectool setProperty "/myJob/report-urls/<myReportName>"/commander/jobs/<jobId>/<w  
orkspaceName>/<artifactDirectoryName>]/<file-path>
```

Example command:

```
ectool setProperty "/myJob/report-urls/Test Report"/commander/jobs/$[/myJob/jobI  
d]/  
$[/myJobStep/workspaceName]/testreport/index.html
```

it will continue to work for non-local workspaces only. It is recommended, however, to change the command to use the newer format.

- **Create a link to any directory:**

To add a link, run the following style command in any job's job step:

```
ectool setProperty "/myJobStep/report-urls/Main Job Workspace"  
"file:///WinStor2/scratch/chron55build/$[/myJob/jobName]"
```

Note: Links to a directory automatically work in Internet Explorer, but if using Firefox, local links are disabled unless the default security policy is modified or an extension is used. Refer to http://kb.mozillazine.org/Links_to_local_pages_don%27t_work for more details.

The "tabbed" Sections

The next section of tabs allows you to select the type of information you want to see. You may see six or fewer tabs, depending on the type of job step you are viewing. See the six tabs illustrated in the following screen example:

Child Steps						
<div>General</div> <div>Diagnostics</div> <div>Parameters</div> <div>Properties</div> <div>Notifiers</div>						
View: All ▾						
Step Name	Log	Status	Elapsed Time	Resource	Actions	
Reserve resource		Success	00:00:25.906	ecbuild-win4	Edit	
Remove old state - windows		Success	00:00:07.844	ecbuild-win4	Edit	
Remove old state - unix		Skipped	00:00:00.000		Edit	
<div>Compile server</div> <div>Compile</div>		Success	00:02:16.203		Edit	
		1403 compiles	00:02:14.078	ecbuild-win4	Edit	
Signal to server tests & install		Success	00:00:02.188	eng	Edit	

Child Steps table

Note: The Child Steps table is available for subprocedure and broadcast steps only.

- Click on a Step Name to go to that step's Job Step Details page.
- Click the Log "icon" to see that job step's log information.
- Status - This column provides the job's step status.
 - Aborted - The step or job was aborted.
 - Canceled - The step did not run because the job was aborted or a previous step was aborted.
 - Error - The command failed or `postp` detected an error in the output.
 - Running - The step is currently running or a step inside the subprocedure call is running.
 - Skipped - The step did not run because the run condition evaluated to "false".
 - Success - The step completed with no errors or warnings.
 - Waiting for Resource or Workspace - The step is waiting to run as soon as a resource or workspace is available.
 - Warning - `postp` detected a warning message.

Note: The Status field can be over-ridden by `postp` to include more specific information. `postp` information may be linked directly to diagnostic information—also available by selecting the Diagnostics tab.

- Click on a Resource name to go to the Edit Resource page.
- Click the **Edit** link in the Action column to go to the Edit Step page.

General table

This section displays basic information including the step ID, step name, create and end times, and whether the step invoked a subprocedure or ran a command.

Child Steps	General	Diagnostics	Parameters	Properties	Notifiers
<p>General</p> <p>Step Id: 2131105</p> <p>Step Name: Windows</p> <p>Subproject: Commander</p> <p>Subprocedure: Master-OnePlatform</p> <p>Create Time: 2008-09-10 05:00:00 PDT</p> <p>Elapsed Time: 01:45:59.472</p> <p>End Time: 2008-09-10 06:48:54 PDT</p> <p>Last Modified: 2008-09-10 06:50:16 PDT</p> <p>Last Modified By: admin</p> <p>Run Condition: true</p> <p>Parallel: 1</p> <p>Diagnostics</p> <p>Exit Code: 0</p> <p>Error Handling: failProcedure</p>					

Diagnostics table

If the job step or child step (for subprocedure and broadcast steps) generated warning or error messages, you will see those messages here. This section contains one or more diagnostics, each of which is a portion of a step's log file that was identified as "interesting" by the step's postprocessor. Typically, each diagnostic relates to an error or warning detected by the postprocessor. All diagnostics for a particular job step appear together.

Parameters table

This section displays parameter values supplied to the top-level procedure when the job step was invoked.

Note: This table is available for steps with child steps only.

Child Steps	General	Diagnostics	Parameters	Properties	Notifiers
Name	Value				
branch	3.0				
cm	lonestar-cm				
dbType	oracle				
platform	windows				
resource	ec-win				
skipServerUnitTests	0				

Properties table

This section displays information computed and/or used by the job step while it is running. After the job step completes, these properties are normally read-only.

- Click on a Property Name to edit that property.
- If a "folder icon" precedes a property name, it denotes a Nested Property Sheet.
- This section also provides **Create Property**, **Create Nested Property**, and **Access Control** links to update or supply additional information for the next time this job step is run. For more information on properties, see the [Properties](#) Help topic.

Child Steps	General	Diagnostics	Parameters	Properties	Notifiers
Create Property Create Nested Sheet Access Control					
Property Name	Value	Description	Actions		
branch	3.0		Delete		
cm	lonestar-cm		Delete		
dbType	oracle		Delete		
platform	windows		Delete		
resource	ec-win		Delete		
skipServerUnitTests	0		Delete		

Notifiers table

If you have designated certain persons or groups to be notified of details for specific job steps, this section displays who receives an email notification, the type of notification, any condition, and a description [if supplied]. If a notifier is available, you can click on its name to view details or make modifications.

Job Step Time and WaitTime Properties Explained

- **start** - The time when this job step began executing.

When the Commander server assigns a resource to run the step, it issues a `<runcommand>` request containing the expanded command body to that resource's agent. The "start" time is when that `<runcommand>` is sent to the agent.

- **finish** - The time when this job step completed.

When a command finishes executing, the Commander agent issues a `<finishcommand>` request containing the exit code and other metadata back to the Commander server. The "finish" time is when the `<finishcommand>` is received by the server.

- **elapsedTime** - The number of milliseconds between the start and end times for the job.

This is exactly "finish" - "start".

- **totalWaitTime** - The sum of resource, workspace, and license wait times for this job step.

The sum of `resourceWaitTime`, `licenseWaitTime`, and `workspaceWaitTime`.

- **waitTime** - The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).

Covers known reasons why a step couldn't run as soon as it became runnable (same as `totalWaitTime`), plus any other possible reason (slow transaction commits, system paging, etc).

- **resourceWaitTime** - The length of time this job step stalled because it could not get a resource. 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.

For subprocedure steps, this is the sum of all child steps.

- **licenseWaitTime** - The length of time this job step had to wait to process because the license limit was reached or exceeded.

For subprocedure steps, this is the sum of all child steps.

- **workspaceWaitTime** - The time this job step had to wait because no workspace was found or available.

For subprocedure steps, this is the sum of all child steps.

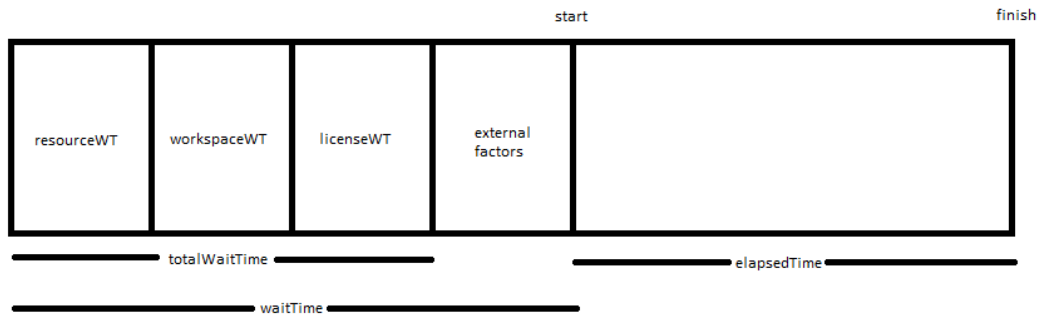
When it is time to schedule the step, Commander checks the license and starts counting time against `licenseWaitTime` until the license is satisfied (e.g., running this step would not exceed the maximum number of licensed hosts). After the step is scheduled, Commander sends a `runCommand` message to the agent. If it fails because Commander can't talk to the agent, Commander accrues time against `resourceWaitTime`. If Commander can talk to the agent, but it has trouble manipulating the workspace (e.g., can't create the workspace dir, map the drive (on Windows), or create files), Commander accrues time against `workspaceWaitTime`.

It is possible for the following to occur:

1. A job step becomes stalls due to a license issue.
2. The license limit is no longer exceeded and Commander tries to run the step but hits a bad resource.

3. When the resource is good, the step stall because of a license again (adding more time to the licenseWaitTime),
4. When the license is satisfied, Commander reaches the resource, and everything runs.

This is why Commander continues cycling through these states until everything is satisfied.



The first 4 blocks (resourceWaitTime, workspaceWaitTime, licenseWaitTime, and external factors) can occur in any order.

External factors could include things such as system paging, slow transaction commits, etc.

Licenses

This page displays all license information known to the ElectricCommander server. Typically, a single license is displayed, which describes the usage to which you are entitled. The server may be licensed by concurrent resources, concurrent users, concurrent steps, or any combination of these licenses.

All Licenses

Click the **Import License** link to import a new license file.

Column descriptions

Column Name	Description / Actions
Company	The name of the company or organization to which the license was granted. If this is not your company, please contact Electric Cloud Customer Support.
Feature	The type of license.
Expiration	The last day when this license is no longer valid. After your license expires (and after any grace period), running jobs will complete, but no new jobs can be launched.
Grace Period (days)	If this field is nonzero, you may continue using Commander for this many days after the expiration date. However, you will be warned frequently during the grace period that your license has expired.
Actions	View License - use this link to see additional details about this license. Delete - click this link to delete this license.

Current Usage section

You will see some or all of the following fields depending on the type of license or licenses you have chosen to use for Commander.

Field Name	Description
Maximum Concurrent Steps	The maximum number of concurrent steps allowed by this license.
Concurrent Steps	The number of concurrent steps running now.

Field Name	Description
Maximum Managed Hosts In Use	The maximum number of unique host names defined in <i>managed</i> resources (non-proxy resources), where steps can run at any time.
Managed Hosts in Use	The number of unique host names defined in <i>managed</i> resources where steps are currently running.
Maximum Proxied Hosts In Use	The maximum number of unique host names defined in <i>proxied</i> resources, where steps can run at any time.
Proxied Hosts in Use	The number of unique host names defined in <i>proxied</i> resources where steps are currently running.
The following fields are specific to a user-based license	
Maximum Concurrent Users	The number of users allowed to use Commander concurrently at any given time.
Active Users	The number of users with an active license.
User IP	The "user@IP" combination that holds an active license.
Expiration	The time when the license will be released.
Last Use	The time when the license was last used.

Three License types

Concurrent Resource License

This Commander license controls the number of unique host names on which steps can be scheduled at any time. Two properties, `maxHosts` and `maxProxiedHosts`, are defined on a license to control the maximum number of managed and proxied hosts that can be in use at the same time. The host names "localhost" and "127.0.0.1" refer to the local agent installed on the Commander server machine, and these host names are not part of the total count for hosts in use.

If two or more steps are running concurrently on the same host, the host name still counts once only towards the license limit. There is no limit to the number of resources pointing to a single host; from a licensing perspective, they all count as a single host. When the host limit is met, any new job step that attempts to run on a host [not already in use] will stay in the "runnable" state until its host is available. Current license usage statistics are displayed at the top of the Resources page in the web interface. If no license is available, jobs cannot be launched. When a license expires, running jobs are allowed to complete, but no new jobs can be launched.

Concurrent User License

If the `maxConcurrentUsers` property is set on the license, its value defines the maximum number of users who can actively use the Commander server. A unique license is defined by the combination of the user's name and the IP address from which the user is making requests. A user can have multiple active sessions (through the web UI and `ectool`) from the same machine while only using a single license.

If a user makes a licensed API call without an active license, the server automatically attempts to assign a license. If a license is available (because the maximum has not been met), the user is assigned a license. If no licenses are available, the user will see a `NoAvailableLicenses` error and be redirected to the licenses page. Currently held licenses and their expiration dates can be viewed by calling the `getLicenseUsage` API or by viewing the Active Users table on the Licenses web page. When a license becomes available, the first user to make a licensed API call will be assigned that license. The only API calls not licensed are `login`, `getLicenseUsage`, `getAdminLicense`, `getServerStatus`, and `getServerInfo`.

When a license is first acquired, its expiration is set to 1 hour in the future. After the initial hour has passed, each time the user makes a licensed API call, the license is extended another 10 minutes (a configurable server setting). When a user's license passes the expiration date, it is released. When the user makes another licensed API call, a license is assigned automatically if one is available, otherwise the `NoAvailableLicenses` error is displayed.

A single administrator license can be used in an emergency. All users who have `execute` permission on the system-level licensing access control list can use the administrator license by calling the `getAdminLicense` API. This license has no expiration as long as all regular licenses are in use, but this license abides by the regular license expiration policy if regular licenses are available.

Concurrent Step License

This Commander license controls the number of unique steps you are allowed to run in the system at the same time (in parallel). The `maxConcurrentSteps` property, defined on a license, controls the maximum number of steps that can be running at the same time.

View License

This page displays all available license details for a single license.

Use the **Delete** link to remove this license.

Information section descriptions

Name	Description
Company Name	The name of the company or organization to which the license was granted. If this is not your company, contact Electric Cloud Customer Support.
Product Name	Product to which this license applies (if it is not "ElectricCommander," it will not apply here).
Feature Name	Product feature to which this license applies: must be "Server" for ElectricCommander licenses.
Maximum Concurrent Steps	The maximum number of concurrent steps allowed by the license.
Maximum Managed Hosts In Use	The maximum number of unique host names defined in managed resources (non-proxy resources), where steps can run at any time.
Maximum Proxied Hosts In Use	This is the maximum number of unique host names defined in proxied resources, where steps can run at any time.
Maximum Concurrent Users	The number of users allowed to use Commander concurrently at any given time.
Expiration Date	The last day when this license is no longer valid. After your license expires (and after any grace period), running jobs will complete, but no new jobs can be launched.
Grace Period	If this field is non-zero, you may continue using Commander for this many days after the expiration date. However, you will be warned frequently during the grace period that your license has expired.

Import License

Use this page to provide a new license for ElectricCommander.

You should have received a license [as a block of text] from Electric Cloud already. If you have not received a license, contact Electric Cloud Customer Support or your sales representative.

- Use a text editor to open the license file.
- Copy the text into the Contents field.
- Click **OK**.

You will see your license information on the Licenses page.

Plugin Manager

A plugin is a collection of one or more features that can be added to ElectricCommander.

- Plugins are delivered as a JAR file containing the features implementation.
- When a plugin is installed, the ElectricCommander server extracts the JAR contents to disk into a configurable plugins directory.
- A plugin has an associated project that can contain procedures and properties required by the implementation.
- A plugin can provide one or more new pages for the web interface and may also provide a configuration page so you can provide any additional information that may be necessary to implement the plugin.

Commander provides default/bundled plugins, installed during the Commander installation. These plugins are "authored" and generally supported by Electric Cloud.

The information Plugin Manager displays across the top of the table:

Tab and field descriptions above the table

Currently Installed - The Plugin Manager default page view displays currently installed plugins. If you install additional Commander plugins from other sources, including those plugins you may create, they will also appear on this page.

Install from File/URL - Select this tab to see the following screen for plugin installation options:

File Install - Use this method to upload and install a local "jar" file from the machine running your web browser. Click **Upload** after specifying the path you need.

URL Install - install a plugin from a location specified by an URL. This location can either be an external web server (using `http://`), or a file on the Commander server host (using `file://`)

Install a plugin from the command-line:

ectool - the command-line tool, contains a full set of commands to perform plugin tasks. For more information on available ectool plugin commands, see the "API Commands / [Plugin Management](#) group" online help topic.

Note: The server has a configurable maximum file upload size that defaults to 50MB. Refer to the server "Maximum upload size" setting to adjust the default.

View Catalog - This page displays all plugins available for installation. If you do not see an Install option in the Actions column, you may not have the necessary access control permissions to install a plugin.

Catalog Install - click on the **Install** action (next to a plugin listed in the catalog) to download and install the plugin.

Category: a drop-down menu to filter the plugin list by a specific category.

Column descriptions

Column Name	Description / Actions
Plugin Label	The name of the plugin. Click on the label of an installed plugin to go to the Project Details page for that plugin.
Author	Identifies the person or organization that created the plugin and optionally provides a link to contact the author.
Version	The plugin version. Promoted plugin versions are also shown in this column. A promoted version number is labeled "(promoted)" immediately after the version number.
Category	The plugin type. Plugins that provide similar types of functionality are grouped into categories.
Description	A short text string that describes the plugin purpose or function.

Column Name	Description / Actions
Actions	<p>This column contains one or more of the following links to perform plugin actions or tasks. Not all plugins have the same task options. For example, some plugins may perform their own "setup" during installation and not require additional configuration information.</p> <p>Note: If you do not see the option you need in the Action column, it may be hidden from your view if you do not have sufficient access control permissions.</p> <p>Possible Actions:</p> <p>Configure - use this link to go to the configuration page for the plugin.</p> <p>Demote - use this link to make this plugin inactive and remove any tabs associated with the plugin.</p> <p>Note: If you re-install a previously demoted plugin, previous values from the demoted version are not copied to the new version you install.</p> <p>Help - if provided, use this link to go to the plugin documentation.</p> <p>Install - Download and install a plugin from the plugins catalog site.</p> <p>Promote - use this link to upgrade the plugin to a new version. Old plugin values are copied to the promoted version.</p> <p>Re-install - use this link to download and reinstall a previously installed plugin.</p> <p>Uninstall - use this link to remove this plugin from your system.</p> <p>Note: the Demote, Promote, Install, Re-install, and Uninstall actions appear only if the user has modify permission to the "plugins" ACL.</p>

Using a plugin

Multiple methods for adding plugin functionality to Commander are available.

- To configure Commander to recognize your SCM plugin, see the [Configuring Commander](#) help topic and select the *Setting Up a Source Control Configuration* section.
 - For additional plugin help, see the [Customizing the Commander UI](#) help topic.
 - For additional help using your SCM plugin with Commander Preflight, Default Tracking, and ElectricSentry, see those help topics: [Preflight Builds](#), [Defect Tracking](#), [ElectricSentry](#)
- To add a subprocedure step using the Edit Step or New Step page in the Commander UI:
 1. Select the Projects tab and choose an existing project or click the **Create Project** link if you need to create a new project.

2. On the Project Details page, select an existing procedure or click the **Create Procedure** link if you need to create additional procedures.
3. On the Procedure Details page, to create a New Step using a plugin, select the **Plugins** link to go to the Choose Step panel.
 - In the left pane, select a plugin category and then choose a plugin.
 - In the right pane, select the step type you want, which will take you to the New Step page to add a new subprocedure step (using your selected plugin).
4. On the New Step page, notice that the plugin you selected is already displayed in the Subprocedure section.
5. Fill-in the Parameters section fields. These fields are specific to the plugin you chose. (These fields are frequently different for each plugin.)

Configuring the plugins directory

For instructions on configuring the plugins directory for your Commander server, remote agents, and/or remote web servers, see the *ElectricCommander Installation Guide*.

Procedure Details

This page displays procedure component tables, including a table for Procedure Steps, Parameters, Email Notifications, and Custom Procedure Properties. A procedure describes a process to execute that consists of one or more step.

Use this page to create steps.

Procedure Steps

This table displays all steps in the procedure [named at the top of the page]. The steps execute in order from top to bottom. Normally only one step executes at a time, but if a group of adjacent steps is marked as "parallel," all of those steps will execute simultaneously.

Important: If you want to change the order of the steps, click on the icon to the left of the step name and drag it up or down to the new step position.

Note: This table displays only a few of the fields for each step. To view (and modify) the complete details for a step, click on the step's name to go to the Edit Step page.

Links at the top of the page

- Use the **Run** link [hovering your mouse over the small down-arrow on the right-side of the link] to choose Run Immediately or Run...
 - Run Immediately - Selecting this option runs the procedure immediately as set.
 - Run... - Selecting this option displays the Run Procedure web page where you may alter any existing parameters for this procedure, or set an existing credential.
- Click the **Access Control** link to go to the Access Control page to add privileges for this procedure.
- Click the **Edit** link to go to the Edit Procedure page so you can make modifications if necessary.
- Use the **Copy** link in the Actions column to make an exact copy of this step.
- Choose one of the New Step links to create the type of step you need.

Creating a new step

The following list describes some of the more common steps you can create.

- **Command** - this link takes you directly to the New Step page to write your own step. A Command(s) text box is available to add a script. This step invokes a `bat`, `cmd`, `shell`, `perl` script, or similar.
- **Subprocedure** - the Choose Subprocedure Step dialog is displayed, allowing you to select a project or plugin from which you want to call an existing procedure to create a step. This step invokes another Commander procedure and will not complete until all subprocedure steps have finished.
- **Plugin** - this link opens the Choose Step panel. The left pane displays plugin categories and the right pane displays available pre-configured steps for your left-pane selection. To quickly find your application/plugin, type your selection name in the Search box. Select the step you want to create and Commander takes you to the New Steps page.
 - On the New Step page, notice that the plugin step you selected is already displayed in the Subprocedure section.

- Fill-in the Parameters section fields. These fields are specific to the plugin you chose. (These fields are different for each plugin.)

Fill-in any remaining fields to complete your step. See the [New Step help](#) topic for more information.

A note about plugins

Plugins are the vehicle Commander uses to integrate source control systems (such as Subversion or Perforce), defect tracking applications (such as JIRA or MKS), reporting functionality, code analysis applications, build applications, and much more. To see the complete list of plugins bundled and installed with Commander, see the [Plugin Manager](#) page (go to Administration > Plugins).

Note: If you do not see the SCM system or defect tracking system you prefer, a Plugin Catalog is available from the [Plugin Manager](#) also. All plugins in the catalog are available for installation into Commander.

Parameters

Each procedure can define parameters, which are values provided to the procedure when it is invoked. Each parameter value is placed in a property associated with the job and the procedure can use the property to control its execution. For example, a parameter might specify a particular branch of a product to build; another parameter might indicate whether unit tests should be run during this build.

When you define a parameter, indicate whether the parameter is required (meaning the procedure will not run unless a value is provided for the parameter). Also, you can provide a default parameter value and a description to help callers understand how to use the parameter.

- Click the **Create Parameter** link to go to the New Parameter page.
- Click on the Parameter Name to edit its contents.

Email Notifiers

Use the **Create On Start Email Notifier** or **Create On Completion Email Notifier** links to go to one of the respective pages to create an email notifier for this procedure.

After creating an email notifier, click on the "bread crumb" link at the top of the page to return to the Procedure Details page.

Custom Procedure Properties

- To add new properties to this procedure, click the **Create Property**, **Create Nested Sheet**, or **Access Control** links.
- Click on the Property Name to edit its contents.
- Nested Property Sheets appear in this table preceded by a folder icon.

Tip

Want to rename this procedure or change its description? Click the **Edit** link at the top of the page.

Procedure - create new or edit existing procedure

To create a new procedure

Fill-in the fields as follows:

Field Name	Description
Name	Supply a name for the procedure. This name must be unique within the project.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Job Name Template	<p>This is the template used to determine the default name of jobs launched from the procedure.</p> <p>For example:</p> <pre> \$[projectName]_\${/increment /myproject/jobCounter}_ \${/timestamp} </pre> <p>Produces a name like:</p> <pre> projectFoo_1234_20140102130321 </pre> <p>Supply any combination of elements to create procedure names more meaningful to you. For example, you could choose to include the build number.</p> <p>Note: Electric Cloud does not recommend using the Commander-generated <code>jobId</code> because it is no longer a human-readable integer so it does not provide any identifiable information and cannot be used as a counter.</p>
Default Resource	Supply a resource name. This name must be unique among other resource names or click Browse to select a resource.
Default Workspace	Supply the name of the workspace or click Browse to select a workspace name.

Field Name	Description
Time Limit	<p>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.</p> <p>The "timer" for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).</p> <p>Specify a number and then use the pull-down menu to select the "time units" of the number you specified.</p>
<i>Impersonation Credential</i>	
Credential Project	Default Credential Project is the current project, or select the other button and Browse for the project name you need.
Credential Name	The name of the credential for the project - you may need to Browse to find this name.

Click **OK** to save the information to create a new procedure.

To edit an existing procedure

Use this page to modify certain overall information about a procedure. Additional information, such as procedure steps and properties, can be viewed and modified on the Procedure Details page. To access this page, click the **Project Name** link on the location path (breadcrumb), then select a Procedure name.

To learn more about a particular field on the form, mouse-over the field to pop-up descriptive text under the mouse or click the **Help** link at the top of the page.

Tip

Want to rename the procedure? Enter a new procedure name in the Name field, click **OK**.

Step - create new or edit existing step


Use this page to define a new step or to modify an existing step. The following information describes the common fields you need to fill-in to create any type of step.

Note: If you chose to create a step using the Plugin link, additional help is available in the Parameters section by clicking the "?" icon. The Parameter section is populated according to type you chose to create. Different step types require different information.

To create a new command or subprocedure step

Fill-in the fields as follows:

Field Name	Description
<i>General section (for all step types)</i>	
Name	Supply a name for the step—must be a unique name within the procedure.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
<i>Command section (for command steps only)</i>	
Command(s)	A script to execute the step functions; passed to the step's shell for execution.
Resource	Supply a unique resource or resource pool name or click Browse to select a resource. If you leave this field blank, the procedure or project provides a default resource. <ul style="list-style-type: none"> The job step must have the execute privilege on the resource or resource pool. If it does not, the job step will be canceled with an access denied error. The resource or resource pool must be enabled. If the resource or pool is not enabled, no error is generated, and the step remains in the runnable state until the pool is enabled.
Postprocessor	If this field remains blank, no postprocessor runs for the step. If this field is not blank, it specifies a command (passed to the step's shell for execution) that analyzes the log file for the step and collects diagnostic information for reporting. <i>For more information</i> , see the Postprocessors Help topic.
<i>Subprocedure section (for subprocedure steps only)</i>	

Field Name	Description
Procedure	Displays the current project and procedure. Click Change to use the popup menu to select a new project/procedure.
Resource	Supply a unique resource name or click Browse to select a resource. If you leave this field blank, the procedure or project provides a default resource.
<p><i>Parameters section (for subprocedure steps only)</i></p> <p>This section expands automatically, supplying the fields you need specifically for the Subprocedure you chose in the previous section. For additional help with field descriptions, click the "?" icon if available.</p> <p>To toggle between the standard view and property view for this section, click the  icon.</p>	
<p><i>Advanced section (for both command and subprocedure steps)</i></p>	

Field Name	Description
Precondition	<p>By default, if this field is blank, the step has no precondition and will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (/myJob/buildInstallerCompleted=1). • Set a precondition in step 4: \$[/myJob/buildInstallerCompleted]
Run Condition	<p>By default, if this field is blank the step will always run. This field accepts "fixed text" or text embedding property references that are evaluated into a logical TRUE or FALSE. A "0" or "false" is interpreted as FALSE. An empty string or any other result is interpreted as TRUE.</p> <p>The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". \$[/javascript (myStep.stepName == myJob.restartStep)]</p>

Field Name	Description
Error handling	<p>This field determines what happens to the procedure if the step fails.</p> <p>The first choice, "procedure continues, but overall status will be error" means an error in the step does not abort the procedure; subsequent steps will run as usual. However, an error will be reflected in the overall outcome of the procedure. If this is the top-level procedure in the job, the job will have an error outcome.</p> <p>If this is a subprocedure, the step invoking the subprocedure will have an error outcome, which still could cause the job to abort, depending on the error handling for that step.</p> <p>The second choice for this field is "abort procedure but allow running steps to continue." In this case, if an error occurs, no new steps are initiated in this procedure except those where "always run" was selected. Any steps currently running are allowed to complete, then the procedure will abort and its outcome will be set to error as described previously.</p> <p>Six error handling options are:</p> <ul style="list-style-type: none"> • <code>failProcedure</code> - The current procedure continues, but the overall status is error (default). • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - Continues as if the step succeeded.
Time Limit	<p>The maximum amount of time the step can execute. If the step exceeds this time, it will be aborted. Specify a number and then use the pull-down menu to select the "time units" of the number you specified.</p>
Run in parallel	<p>If this box is "checked," this step will run concurrently with any adjacent steps marked as parallel. If this box is not checked, the step will NOT run until all previous steps in the procedure have completed, and it will complete before any following steps execute.</p>

Field Name	Description
Always run step	If a procedure is aborted (either because of an error in a step or because the job was aborted), under normal conditions no new steps will start in the procedure. However, if this box is "checked," the step will run even if the procedure is aborted. This facility is most commonly used for steps at the end of a job that generate reports or perform cleanup operations. Note: Remember that the Run Condition field is still evaluated and may cause the step to be skipped even if this box is checked.
Retain Exclusive	Select one of the following options: <ul style="list-style-type: none"> • None - the "default", which does not retain a resource. • Job - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources—if this resource meets the needs of the steps and its step limit is not exceeded. • Step - keeps the resource for the duration of the step. • Call - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.
Workspace	Use this field to define a workspace for the step to use. You can either type-in a workspace name or click Browse to select a workspace. If you leave this field blank, the workspace is determined by the procedure or project.
<i>These "Advanced" fields are used for command steps only</i>	
Broadcast	"Checking" this box means replicate the step to execute (in parallel) on each of the specified resources (for example, if a resource pool is specified, run the step on each resource in the pool).
Release Exclusive	Select one of the following options: <ul style="list-style-type: none"> • None - the "default" - no action if the resource was not previously marked as "retain". • Release - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs. • Release to job - allows a step to promote a Step exclusive resource to a Job exclusive resource.

Field Name	Description
Shell	<p>This shell will be used to execute the step's commands on a resource. For example, using <code>sh</code> or <code>cmd /c</code>, the agent saves the command block to a temporary file with a <code>.cmd</code> extension and runs it:</p> <pre>sh foo.cmd or cmd /c foo.cmd</pre> <p>If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:</p> <p>Windows: <code>cmd /q /c "{0}.cmd"</code> UNIX: <code>sh -e "{0}.cmd"</code></p> <p>When you specify a shell (in the step or resource), and omit the <code>cmd-file</code> marker, the agent notices the omission and takes the correct action. For example: a user specifies <code>sh -x</code>. The agent converts this to <code>sh -x "{0}.cmd"</code></p> <p>Two alternate forms of shell syntax where Commander uses a "marker," <code>{0}</code>, as a placeholder for the command file argument:</p> <ul style="list-style-type: none"> • <myShell> {0} <potential extra shell args> In this example, the command file is not meant to be the last argument in the final command line. For example, <code>mysql -e "source {0}"</code> This shell example runs the <code>mysql</code> command against this step's command containing <code>sql</code>. • <myShell> {0} <.file extension> <potential extra shell args> In this example, the shell requires the command file to end in an extension other than <code>.cmd</code>. For example, <code>powershell "& '{0}.ps1'"</code> This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands. <p>Notes:</p> <ul style="list-style-type: none"> • When the agent parses the shell, it will parse the extension as everything after <code>{0}</code> . until it sees a space or non-alphanumeric character. • If your script uses International characters (non-ascii), add the following block to the top of your <code>ec-perl</code> command block: <pre>use utf8; ElectricCommander::initEncodings</pre>

Field Name	Description
Working Directory	The directory where the commands for this step execute. A relative name is interpreted relative to the root directory for the job workspace. If you leave this field blank, the step's working directory will be the top-level directory in the job workspace. Note: If this step will run on a proxy resource, this directory must exist on the proxy target. The step will run in this directory on the proxy target.
Log File	The step's log file name, specified relative to the root directory in the job workspace. If you leave this field blank, ElectricCommander picks a unique name for the log file based on the step name.
<i>Impersonation section (for both command and subprocedure steps)</i>	
Credential Project	By default, the Current project is used, or click Browse and select a different project.
Credential Name	If you select a credential in this field, the step runs under the account from that credential. This field is usually blank, which means a default credential is used. For more detailed information about credentials, see the Credentials and User Impersonation Help topic.

Click **OK** to create the step.

To edit an existing command or subprocedure step

You can modify any of your previously entered step information and add information to one or all of the four additional sections.

Attached Credentials

Click the **Attach Credential** link to retrieve an existing credential for this step. If you need to create a new attached credential, return to the Project Details page for this step.

Attached Parameter Credentials

Click the **Attach Parameter Credential** link to retrieve an existing parameter credential for this step.

Email Notifiers

Choose **Create On Start Email Notifier** or **Create On Completion Email Notifier** to go the respective page to create the email notifier you need.

Custom Step Properties

Choose from the **Create Property**, **Create Nested Property**, or **Access Control** links to add a custom property to this step. Each Property pop-up box has its own Help topic or go to the main [Properties](#) Help

topic for more information. For more information about Access Control, go to the main [Access Control](#) Help topic.

Click **OK** to save your modified step information.

Publish Artifact Version Step

Use the EC-Artifact plugin to create a step to publish a new artifact version.

The following parameters are available to create this step:

Field Name	Description
Artifact	Artifact name, in the form <code><groupId>:<artifactKey></code> . If the artifact does not exist, it will be created if this procedure's launching user or this project principal has the required permissions.
Version	A full version string takes the form: <code><major>.<minor>.<patch>-<qualifier>-<buildNumber></code> . The version specification must be unique across all of this artifact's versions.
Repository	Name of the repository where this new artifact version will be published.
Enable Compression	Check this box to compress the artifact version before it is stored in the repository.
From Directory	Name of the directory in the job's workspace containing files that comprise the artifact version to be published. If not specified, the entire workspace is used.
Include Patterns	List file "include" patterns one pattern per line, to limit which files are published. If no patterns are specified, all files are included.
Exclude Patterns	List file "exclude" patterns one pattern per line, to limit which files are published. If no patterns are specified, no files are excluded.
Dependent Artifact Versions	List dependent artifact versions one per line, each in the form <code><groupId>:<artifactKey>:<versionRange></code> . All dependent artifact versions must exist for this artifact version to be retrievable. When this artifact version is successfully retrieved, its dependent artifact versions are retrieved also.

Retrieve Artifact Version Step

Use the EC-Artifact plugin to create a step to retrieve an artifact version.

The following parameters are available to create this step:

Field Name	Description
Artifact	Artifact name, in the form <groupId>:<artifactKey>.
Version	Select the latest version (Latest), exact version of the artifact (Exact), or a version range (Range). Version is in the form: <major>.<minor>.<patch>-<qualifier>-<buildNumber>
Retrieve to directory	If you want to retrieve this artifact version to a specific directory location, select the checkbox, then specify the full path to the directory where you want to retrieve this artifact version.
Overwrite	Use the drop-down menu to select one of the following options: <ul style="list-style-type: none">• <code>true</code> - deletes previous content in the directory and replaces the content with your new version.• <code>false</code> - (existing behavior) if the directory does not exist, one will be created and filled with the artifact's content. If the directory exists, a new directory is created with a unique name and the artifact contents is supplied there.• <code>update</code> - this is similar to a merge operation—two artifact versions can be moved into the same directory, but individual files with the same name will be overwritten.

Field Name	Description
Retrieved Artifact Location Property	<p>Name or property sheet path used by the step to create a property sheet. This property sheet stores information about the retrieved artifact version(s) as XML in the <code>artifactVersionXML</code> child property, and the file system location of the retrieved artifact version in the <code>cacheLocation</code> child property.</p> <p>The initial value of this field includes <code>[\$assignedResourceName]</code> as one of the levels in the property path because different resources may have different cache directories, and you do not want a retrieval on one resource to clobber the data regarding a retrieval on a different resource.</p> <p>Typically, a step that needs to use files from an artifact retrieved by this step will specify it in a Perl script similar to this:</p> <pre> my \$cmdr = new ElectricCommander(); my \$xpath = \$cmdr->getProperty ("/myJob/retrieveArtifactVersions/[\$assignedResourceName] /MyGrp:MyKey/cacheLocation"); my \$retrieveDir = \$xpath->findvalue("//value")->value (); system("java", "-cp", "\$retrieveDir/lib/bar.jar", "Bar"); </pre> <p>This example shows how to retrieve the location of the retrieved <code>MyGrp:MyKey</code> artifact version on the resource, and add a file (<code>bar.jar</code>) in its <code>lib</code> directory to classpath, so the <code>Bar</code> class can be run.</p>
Filters	Supply search filters, one per line, applicable to querying the Commander database for the artifact version to retrieve.

Artifact Retrieval Dependency Order Explained

The order of dependencies registered for an artifact version are significant.

Consider this scenario:

- A depends on B (any version) and C [1.0, 2.0]
- B depends on C (any version)
- C versions 1.0, 2.0, and 3.0 exist

When retrieving A, the dependency algorithm evaluates B first. The algorithm finds that the max version of B depends on any version of C, so the algorithm looks for max version C and finds C 3.0. Because this chain is satisfied, the algorithm returns to A and evaluates its next dependency "C [1.0, 2.0)". This results in matching C 1.0.

The returned artifacts are: A, B, C 1.0, and C 3.0.

Consider if the A dependency is changed to:

- A depends on C [1.0, 2.0), B (any version)

The algorithm will choose C 1.0 first. Then the algorithm evaluates B, determines that its "C (any version)" is satisfiable by the already chosen C 1.0.

The returned artifacts are A, B, and C 1.0.

Note: In the version range syntax [] indicates inclusive, () indicates exclusive.

Send Email Step

Use the EC-SendEmail plugin to create a step to send email notifications.

The following parameters are available to create this step:

Parameter Name	Description
Email Configuration	The name of the email configuration to use. If no configuration is specified, the configuration named "default" is used.
To	List the "To" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
CC	List the "CC" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
BCC	List the "BCC" recipients for the email message, one per line. A recipient can be a user or group name or a complete email address.
Subject	The subject line for the email message.
Message	The body of the email message or a workspace file containing the body. If both a plain text and an HTML message are provided, both values are sent as alternates in a multipart message. If a raw message is provided, both values are sent as alternates in a multipart message. If a raw message is provided, the value should be a properly formatted RFC822 message.
Advanced options	
Multipart Mode	The multipart mode of the email message. Defaults to "none" unless there are multiple parts, in which case it defaults to "mixedRelated".
Header(s)	One or more RFC822 email header lines (for example: "reply-to: user@host.com").
Attachment(s)	One or more files from the job's workspace to send as attachments. The filename extension is examined to determine the content-type.
In-line Attachment(s)	One or more inline attachments specified as a <code>contentId</code> and a workspace filename. The filename extension is examined to determine the content-type.

New Extract Preflight Sources Step

Choose one of the Preflight plugins to create a step to retrieve an artifact version. There are numerous source code management (SCM) plugins. For example, Accurev, Bazaar, CVS, ClearCase, Git, Mercurial, Perforce, SVN, StarTeam, Team Foundation Server, and Vault.

Supply the following information:

Field Name	Description
Step Name	Supply a unique name for your preflight step. You can use any name of your choice.
SCM Configuration	Use the pull-down menu to select a source control configuration. If a configuration for your source control type does not appear in the drop-down menu, select the Administration > Source Control tabs to configure your source control system.
Destination Directory	This field appears after specifying your SCM from the pull-down menu. If this field does not specify "Required", you do not need to supply this information. The Destination Directory is a path relative to the job's workspace, where the source tree will be created.

Click **Submit** after supplying your information.

After clicking **Submit**, the Edit Step page is displayed.

Use the Edit Step page to specify additional information for your preflight step. Also, you can attach a credential, create an email notifier, add custom properties, and more to your step. For assistance using the Edit Step page, access the **Help** link in the upper right corner of the page.

For more information about preflight builds, see the [Preflight Builds](#) help topic.

Parameter - create new or edit existing parameter

To create a new parameter

Fill-in the fields as follows:

Field Name	Description / Action
Name	Supply a unique name to specify the parameter when a procedure or workflow is invoked.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Type - Select the parameter type from the drop-down menu. The following "types" are available.	
Text entry	Allows a short text entry in the Default Value field.
Text area	Expands the Default Value field to allow adding a longer script-type entry.

Field Name	Description / Action
Dropdown menu	<p>Creates a drop-down menu from which to select a value when the parameter is presented.</p> <ul style="list-style-type: none"> • Enter options - Click + Add Option to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected. • Load options from list - Enter a pipe-separated list of options (for example, <code>foo bar baz</code>). The text and value for the options will be the same. • Load options from property sheet - Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format: <ul style="list-style-type: none"> • An <code>optionCount</code> property must exist whose value is the number of options. • For each option, create a nested property sheet called <code>optionN</code>, where N is the option number, starting with 1. • In each nested sheet, create two properties - <code>text</code> and <code>value</code>. The value of the property <code>text</code> will be displayed in the menu. The value of the property <code>value</code> is the parameter value if that option is selected. • If <code>optionCount</code> is set to 3, you must create three nested sheets - <code>option1</code>, <code>option2</code>, and <code>option3</code>.
Radio selector	<p>Creates "radio" buttons to select an entry when the parameter is presented.</p> <ul style="list-style-type: none"> • Enter options - Click + Add Option to add a new row. Type-in the text and value for each option. The text is what will be displayed in the menu, and the value is the parameter value if that option is selected. • Load options from list - Enter a pipe-separated list of options (for example, <code>foo bar baz</code>). The text and value for the options will be the same. • Load options from property sheet - Enter the path to property sheet that contains options for the parameter. The property sheet must be created in a specific format: <ul style="list-style-type: none"> • An <code>optionCount</code> property must exist whose value is the number of options. • For each option, create a nested property sheet called <code>optionN</code>, where N is the option number, starting with 1. • In each nested sheet, create two properties - <code>text</code> and <code>value</code>. The value of the property <code>text</code> will be displayed in the menu. The value of the property <code>value</code> is the parameter value if that option is selected. • If <code>optionCount</code> is set to 3, you must create three nested sheets - <code>option1</code>, <code>option2</code>, and <code>option3</code>.

Field Name	Description / Action
Checkbox	<p>Creates a checkbox for a value to select (or not) when the parameter is presented.</p> <ul style="list-style-type: none"> Value when unchecked - The value of the parameter when the checkbox is unchecked. Value when checked - The value of the checkbox when the checkbox is selected. Initially checked - Whether or not the checkbox should be checked initially. If true, set the "Default value" to match the "Value when checked". If false, set the "Default value" to match the "Value when unchecked".
Credential	<p>Selecting this option requires the user to specify a user name and password to use this parameter at run time.</p> <p>Note: Credential parameters are not supported on state definitions.</p>
Project	<p>Creates a project selector field on the Run Procedure page to choose a different project name where the parameter needs to find additional information.</p>
Saved Filter	<p>Use this option to reference any saved filters, which is a passing the filter into the procedure at run time (primarily used for reports). Saved Filters are stored as a property in your chosen project.</p>
Default Value	<p>You can specify a default value to assign to the parameter if no explicit value is provided.</p>
Required?	<p>Click the checkbox to select the parameter as "required." If the parameter is required, the procedure or workflow will not run without supplying a value for that parameter. In this case, the default value is ignored. After the procedure or workflow begins execution, the parameter value is available in the job or workflow's property sheet with the same name as the parameter.</p>
Defer Expansion?	<p>Click the checkbox to set the parameter expansion to be "deferred." Deferred means the parameter value will not be expanded when the procedure call is expanded, but can be expanded from a command step instead.</p>

Click **OK** after filling-in the fields.

To edit an existing parameter

You may change any information or add new information as necessary.

For example: Want to rename a parameter? Enter a new parameter name in the Name field, click **OK**.

For more information on property sheets, see the [Properties](#) Help topic.

Projects

This page displays all projects available on this ElectricCommander server— the projects you create, Commander-supplied default projects, and Commander solutions. From this page you can navigate into projects to view its procedures, steps, schedules, credentials, workflows, and other components.

A project is a top-level container within Commander. Most information about software production processes, such as procedures, schedules, jobs, and workflows are contained within a project.

- The drop-down menu (at the top of the table) displays all "tags" you previously defined to mark, label, or group related or similar type projects. When you select a tag, the project list changes to display only the projects with that tag.

To add a tag:

- Click the **Edit** link for the project you want to tag.
- On the Edit Project page, supply a tag name in the Tags field.
- After creating a tag, the tag name will appear in the drop-down menu.

Note: For more information on "tagging" projects, see the [Edit Project](#) help topic.

- To create a new project - click the **Create Project** link.
- **Edit, Copy, or Delete** an existing project - click a corresponding link in the Actions column.

Note: If deleting a project, the "background deleter" will mark the project for deletion, but the deletion process may not start until the server completes other tasks already queued or in progress. Deleting a project could take considerable time depending on the size of your project. For example, a project containing hundreds of jobs, steps, and other objects will take longer to delete than a simple project with 3 procedures and 10 steps.

- To see project details - click on a project name in the Project column to go to the Project Details page.

Projects have two purposes

First, projects allow you to create separate work areas for different purposes or groups of people so they do not interfere with each other. For example, different projects can reuse the same names internally without conflict, and each project has its own access control that determines who can use and modify the project. 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.

Second, projects simplify sharing. You can create library projects containing shared procedures and invoke these procedures from other projects. After creating a library project, you can easily copy it to other Commander servers to create uniform processes across your organization.

ElectricCommander includes default projects

Currently, Commander includes the following 4 projects:

- EC-Utilities - this project contains procedures to perform basic Commander tasks. Also, these procedures can be used as examples - copy a procedure to another project and then modify it for your purpose. **Note:** By default, only the "admin" user has execute privileges on the EC-Utilities project. The admin user can enable privileges for additional users or groups.
- EC-Examples - this project contains templates for procedures that perform basic Commander tasks.
- Electric Cloud - this project contains procedures to manage ElectricSentry, specifically the Sentry Monitor for schedules and also contains Commander global reports.

Important notes:

When you create project names, do not use "Electric Cloud" or any other Commander-supplied project name. Duplicating a project name creates a conflict. Also, the "EC-" prefix is reserved for ElectricCommander-supplied project names.

Procedures within each project [above] are maintained by Electric Cloud.

ElectricCommander solutions

Commander solutions are developed in the field by Electric Cloud SEs—our field team has the best experience in applying Commander "best practices" to solve real problems. In addition, Commander solutions can be contributed by Commander users from any customer site.

Potentially, any available solution can be considered a new Commander feature "waiting" to be included in the product if that solution gains wide, general appeal within the Commander customer base. However, because solutions are not part of the Commander product, and not developed by the Commander Engineering team, they are not supported by the Electric Cloud Technical Support team. If and when a solution becomes an integral part of the product, that solution will then have full Technical and Engineering support like any other Commander feature.

Currently, Commander solutions are posted to the Electric Cloud Forum Site at <http://forums.electric-cloud.com/ecforums/> - each solution, accompanied by its own installation instructions and help file, can be downloaded from the forum site. If you have a solution you would like to contribute, the Electric Cloud / ElectricCommander Forum is where you need to post it.

When you install a solution on your Commander server, it will be displayed on the Projects web page as a project.

Installing a Commander Solution

Add new or upgrade existing Commander solutions by running the built-in setup script:

1. Download the Commander <solution name>.jar file from the Electric Cloud Community Forums web site.
For this example, the "solution name" is `Test`.
2. Save your download to a temporary directory on the system that hosts your Commander server.
3. Unpack the jar file in the temporary directory—the file creates a directory named `Test`.
 - If Java JDK is installed, use the included jar utility to unpack the .jar file: `%jar xf Test.jar`
 - For Linux, use any unzip utility, for example: `unzip Test.jar`

- For Windows, use Windows Explorer to access the jar contents if you rename the file to `Test.zip`
4. Run the included `setup.pl` script in the jar through the `ec-perl` interpreter (installed in the ElectricCommander `bin` directory). You must have "admin" or equivalent privileges to run the setup script.

```
ectool login admin
%cd <test>
%ec-perl setup.pl
```

Project - create new or edit existing project

To create a new project

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique project name. You may want your project names to reflect the work groups or teams that will be using them. For example, you might set project names based on the products they support.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Default Resource	<p>Supply a resource name or select Browse to find one. If you have not set up resource names or locations yet, click the Resources tab, then the Resources subtab to see the Resources page.</p> <p>Note: This field is optional. If you specify a resource here, it will be used as the default for all jobs that run under the project. This is a convenient way to use a single resource for an entire project: you can change the resource merely by changing the value here. There are several other places you can specify resources, such as procedures and individual job steps: if you specify a resource in another location, it will take precedence over the one specified here.</p>
Default Workspace	<p>Supply a workspace name or select Browse to find one. If you have not set up workspace names or locations yet, click the Resources tab, then the Workspaces subtab to see the Workspaces page.</p> <p>Note: This field is optional. If you specify a workspace here, it will be used as the default for all jobs that run under the project. This is a convenient way to use a single workspace for an entire project: you can change the workspace merely by changing the value here. There are several other places you can specify workspaces, such as procedures and individual job steps: if you specify a workspace in another location, it will take precedence over the one specified here.</p>

Click **OK** to save the information you entered.

Your new project name will appear on the Projects page.

To edit an existing project

- Want to rename the project? Type a new name in the project Name field and click **OK**.
- You may modify the description or type a text description if this field is blank.
- You can modify any previously entered information.
- If you would like a particular credential to be used (by default) for all jobs that run under this project, select it here. If you leave this field blank, credential information from other sources is used.

Tags - (optional) Supply a tag if you want to categorize or "mark" this project to identify its relationship to one or more other projects or groups. For example, you might want to tag a group of projects as "production" or "workflow", or you might want to use your name so you can quickly sort the project list to see only those projects that are useful to you.

- The tag name you supply in this field is displayed in the drop-down menu on the Projects page. When you select a specific tag from the drop-down menu, you see only the projects with that tag.
- Assigning a tag creates an "ec_tag" property that can be seen (with its value - the tag name you chose) after selecting the Properties subtab from the Project Details page,
- Do not use "spaces" in tag names. For example, if you want to mark a project with the tag name, "Eng Team A", you need to specify, "eng_team_a".
- Do not use symbols in a tag name—results are unpredictable.
- If a project is related to several project categories, you can supply a space-delineated list to add numerous tags at the same time.
Type a new name in the project Name field and click **OK**.
- To delete a tag, select the text and use your Delete key.

Impersonation Credential

Credential Project - The default Credential Project is the current project, or select the other button and Browse for the project name you need.

Credential Name - This is the name of the credential for the project - you may need to Browse to find this name.

Click **OK** after entering new or updated information.

For more information, see the [Credentials and User Impersonation](#) Help topic.

Project Details

This page displays project components, including procedures, workflow specifications, jobs, schedules, any credentials, properties, and reports.

Links and actions at the top of the page

- **Edit** - use this link to go to the Edit Project page.
- **Access Control** - use this link to go the Access Control page for this project.
- The "star" icon allows you to save this job information to your Home page.
- The "curved arrow" icon returns you to the Projects page.
- **Pagination** - Use the "previous" and "next" arrow icons to view the previous or next project. The numbers between the arrow icons display the number of projects you can view and the first number indicates which project [in the list] you are viewing.

The "tabbed" sections

The tabs at the top of the table allow you to select the type of information you want to see. See the 8 tabs illustrated in the following screen example:

Project Details – Electric Cloud

Edit Access Control ☆ ↺ 22/44 ↻

Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
------------	----------------------	------	-----------	-----------	-------------	------------	---------

Create Procedure

Procedure	Description	Resource	Create Date	Actions
runReports	Runs batch reports.	local	2011-09-13 16:43:10 PDT	Run ▾ Edit Copy Delete

The Project Details page opens with the Procedures tab highlighted, so the first table you see is the Procedures table.

Procedures tab

The following links and actions are available in the Procedures table:

- Click the **Create Procedure** link (at the top of the table) to go to the New Procedure page to create another procedure.
- Click on a procedure name (first column) to go to the Procedure Details page for that procedure.
- Action column - click on any of the available links (**Run, Edit, Copy, or Delete**) to perform that function for the procedure listed in that row.

Note: The Run link provides two choices when you hover the mouse over the down-arrow. Select **Run Immediately** or **Run...**

- **Run Immediately** - This option runs the procedure "as-is" immediately.
- **Run...** - This option produces the Run Procedure web page where you can modify any existing parameters for this procedure, or set an existing credential.

Workflow Definitions tab

This tab provides a table listing all defined workflow definitions and includes the following functionality:

- Click the **Create Workflow Definition** link to go to the New Workflow Definition page to define additional definitions.
- Name column - Click on a workflow definition name to go to the Workflow Definition Details page (for that workflow definition).
- Action column
 - **Run** - use the down-arrow to select **Run...** or **Run Immediately**
Selecting **Run...** takes you to the Run Workflow page where you can set the starting state for the workflow.
Selecting **Run Immediately** runs the workflow "as-is", immediately.
 - **Copy** - use this link to make an exact copy of an existing workflow definition.
 - **Edit** - use this link to edit an existing workflow definition or just to change the name of the copy you created.
 - **Delete** - deletes an existing workflow definition on the same row.

The following screen example illustrates the Workflow Definitions table:

Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
Create Workflow Definition							
Name	Description	Actions					
BTD	build test deploy	Run	Copy	Edit	Delete		

Jobs tab

This tab provides a project-specific job table. Selecting the Jobs tab on the Project Details page displays only those jobs for the project you selected.

This table is a subset of the main Jobs tab that displays **all** jobs, regardless of the project of origin, and all Jobs page functionality is basically the same on either Jobs tab.

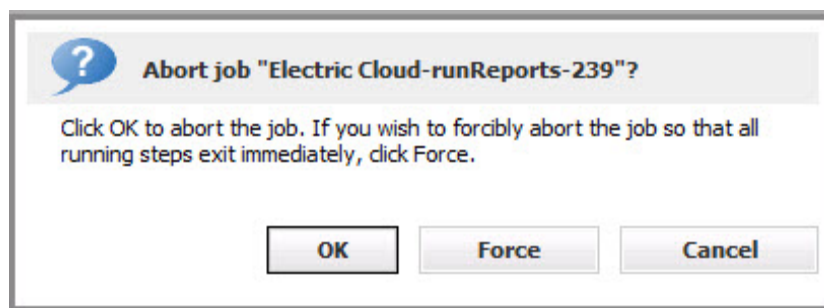
Project Details – Electric Cloud							
Edit Access Control 5/13							
Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
Job	Status	Priority	Procedure	Launched By	Elapsed Time	Start Time	Actions
Electric Cloud-ElectricSentry-8	Success	normal	ECSCM-1.1.0.34932:ElectricSentry	ECSCM-SentryMonitor	00:00:08.897	2010-11-02 15:55:00 PDT	Delete

The following links and actions are available on this Jobs page:

- Job - click on Job to sort the column alphabetically or click on a job name to go to that job's Job Details page.
- Status - click on Status to sort this column by Running, Success, Warning, Error, or Aborted.
- Priority - displays the job's priority set by a "run procedure" command or by the job's schedule.

- Procedure - click on a Procedure name to go to the Procedure Details page for that procedure.
- Launched By - click on a name in this column to go to the page for the schedule (Edit Schedule page) or the user that ran the job.
- Elapsed Time - click Elapsed Time to sort the time values from longest to shortest time, or the reverse.
- Start Time - click Start Time to sort this column from the start time of the first job to the start time of the most recent job, or the reverse.
- Actions - use this column to Abort a running job or Delete a completed job. If your job has finished running, you will not see the Abort link.
 - Aborting a job requires the execute privilege on the job—not just the modify privilege.

When you click **Abort**, the following dialog box appears:



Select either **OK** or **Force** to abort the job, or **Cancel** if you change your mind.

Note: Deleting a job on this page causes removal of job information from the Commander database, but information in the job's on-disk workspace area is not affected. You must delete workspace information manually.

Workflows tab

The Workflows tab provides the following functionality:

- Name column - Click on a workflow name to go to its Workflow Details page.
- State column - Click on a state name to go to its State Details page.
- Modify Time - This time value represents the most recent workflow activity.
- Workflow Definition column - Click on a workflow definition name to go to the Workflow Definition Details page.
- Completed column - a "checkmark" in this column identifies this workflow as completed.
- Actions column - Click the Delete link to remove the workflow in that row.

The following screen example illustrates the Workflows table:

Procedures

Workflow Definitions

Jobs

Workflows

Schedules

Credentials

Properties

Reports

Name	State	Modify Time	Workflow Definition	Completed	Actions
template-1205	stable	2010-12-05 22:17:57 PST	workflow-template		Delete
template-57	building	2010-11-29 16:02:08 PST	workflow-template		Delete
template-99	wip	2010-11-24 10:44:41 PST	workflow-template		Delete
template-959	stable	2010-11-23 09:57:16 PST	workflow-template		Delete
template-958	stable	2010-11-23 09:02:01 PST	workflow-template		Delete
template-957	stable	2010-11-22 15:23:53 PST	workflow-template	✓	Delete
template-954	building	2010-11-22 11:16:36 PST	workflow-template		Delete
template-98	stable	2010-11-19 12:15:14 PST	workflow-template		Delete
template-97	stable	2010-11-19 10:37:18 PST	workflow-template	✓	Delete

For more information about workflows, see the [Workflow](#) help topic.

Schedules tab

The Schedules tab provides the following functionality:

- To create a new schedule for this project, click either the Schedule link to create a new standard schedule to run at specific and/or days or the CI Configuration link to setup a new continuous integration schedule.
 - Choosing "Schedule" displays the New Schedule page.
 - Choosing "CI Configuration" displays the New CI Configuration page.

Table column descriptions

- To edit an existing schedule, click on the schedule name (first column).
- Select the checkbox in the Enabled column to enable or disable the schedule in that row.
- The Priority column displays the job's priority you selected while creating the schedule for this procedure.
- Action column links -
 - **Run** - runs the schedule with the permissions of the logged in user.
For example, if this schedule is timed to run every day at 11:00 pm, you might decide to run the schedule at 6:00 pm on a particular Thursday, without changing the regular schedule settings. However, if you do not have the appropriate permissions to run this schedule, it will fail.
 - **Copy** - makes a copy of the schedule on that row.
 - **Delete** - deletes the schedule in that row.

The following screen example illustrates the Schedules table:

Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
Create: Schedule CI Configuration							
Schedule	Enabled	Procedure	Priority	Description	When to run	Time Zone	Actions
Sentry-MainBuild	<input type="checkbox"/>	Basic Build	normal		Run at 00:00.	America/Los_Angeles	Run Copy Delete

For information on creating a standard schedule, see the [Schedule - create new or edit existing schedule](#) help topic.

For information about CI Manager and adding a configuration, click the Home tab, the Continuous Integration subtab, then click the Help link in the upper-right corner of the page.

Credentials tab

The Credentials tab provides the following functionality:

- To create a new credential for this project, click the **Create Credential** link at the top of the table.
- To edit an existing credential, click on the credential name (first column).
- Action column - the Delete link deletes the credential on that row.

The following screen example illustrates the Credentials table:

Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
							Create Credential
Credential	User	Description	Actions				
Cloud User	AJ Cloud	primary Cloud user	Delete				
Cloud User2	IT Cloud	backup Cloud user	Delete				

For information about credentials and access control, see the [Credentials and User Impersonation](#) or [Access Control](#) help topics.

Properties tab

The Properties tab provides the following functionality:

- To create a new property for this project, click the **Create Property** or **Create Nested Sheet** link.
- To view or change privileges on the property sheet, click the **Access Control** link.
- To edit an existing property, click on the property name (first column).
- Action column - the Delete link deletes the property on that row.

The following screen example illustrates the Properties table:

Procedures	Workflow Definitions	Jobs	Workflows	Schedules	Credentials	Properties	Reports
							Create Property Create Nested Sheet Access Control
Property Name	Value	Description	Actions				
<input type="checkbox"/> ec_dashboardReports			Edit Delete				
<input type="checkbox"/> ec_savedSearches			Edit Delete				

For more information about properties, see the [Properties](#) help topic.

Reports tab

Selecting this subtab yields different results, depending on whether or not you have installed the additional Commander-supplied optional reports or have created custom reports for the specific information you need. Generally, if you have not yet configured additional reports, you will see the three installed, default Commander

reports (Cross Project Summary, Daily Summary, and Resource Summary) as illustrated in the screen example below.

- When a report runs, the report "folder" is populated with the completed report. Reports are listed by date and time.
- Each individual report "date/time" entry is a link to see the full report.
- To create a new report, click the **Create Report** link to go to the web page to create a Multiple Series, Category, Count Over Time, or Procedure Usage report.

The following screen example illustrates the Reports page:



For more information about reports, see the [Reports](#) help topic series.

Tips

Want to rename this project?

Click the **Edit** link at the top of this page and enter a new name.

Want to rename a procedure in this project?

Click the **Edit** link for the procedure (on this page) and enter a new name.

Run Procedure

Using the Run Procedure page

- You may see a message informing you that no parameters were defined for this procedure. If this is the case, click **Run** to run the procedure.
- If parameters were defined for this procedure, you will see specific fields based on those parameters. You may make changes to any field you choose if you want to run this procedure under different parameters at this time. However, any changes you make are specific to this run procedure only. Your changes on this web page will not affect previously defined parameters for this procedure. Click **Run** to run the procedure.
- You may see a "Required" field. To run the procedure, you must fill-in the required parameter previously defined on the Procedure Details page.

Click **Run** to run the procedure.

Advanced section

Priority:

Use the drop-down menu to select the priority for this run procedure. Available priorities are: low, normal (default), high, or highest.

- You can select the job's priority here on the Run Procedure web page or when you schedule the procedure to run at a regular time or interval. When a job is launched, its priority cannot be changed.
- 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.
- If using ectool, the priority can be set by passing "`--priority=<low|normal|high|highest>`" to a `runProcedure`, `createSchedule`, or `modifySchedule` operation.

Note: To raise the job priority level above "normal," the user who launches the procedure must have execute permission on the system-level access control list for priorities.

Impersonation:

Select one of the following:

- **Use pre-defined credential** - (Default option) Use this option if you have not defined a credential for this procedure.
ElectricCommander looks for a credential first in the top-level procedure and then in its project. If it finds a credential, it uses the login from that credential as the default for the job. If there is no credential in either place, by default, the job's steps run as the user under which the agent for the step is running.
- **Use specific credential** - If you select this option, more fields are displayed for you to choose a particular credential and supply its name from those defined for the project containing the procedure. The login user from that credential will be used as the default for the job.

- **Use a specific user** - If you select this option, more fields are displayed to enter a User Name and Password. The specified login is used as the default for the job.

Click **Run** to run the procedure.

For more information on credentials and user impersonation, see the [Credentials and User Impersonation](#) Help topic.


Schedule - create new or edit existing schedule

Use this page to create or edit a standard schedule that specifies a set of times and days when a particular procedure should run. If you need to create a continuous build integration instead, see the Help information available from the Continuous Integration Dashboard web page.

For any schedule, click the "star" icon to add this schedule to the Shortcut section on your Home page.

To create a new standard schedule

Fill-in the fields as follows:

Field Name	Description
General section	
Name	Supply a Schedule name. This must be a unique name among other schedule names in this project.
Procedure	Displays the current project and procedure. Click Change to use the popup to select a new project/procedure.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Parameters section	
If the procedure has parameters, fill-in their values in this section. The procedure cannot execute unless all required parameters are provided. If you move the mouse over a field for a particular parameter, inline help displays the description of that parameter if one was provided in the procedure's definition.	
	Click this icon to displays fields to add one or more parameters.
Frequency section	
Run at ???	This is a summary section: As you make selections for the following Days and Repetition sections, a summary of what you have selected appears here.

Field Name	Description
Days	<p>Select weekdays, month days, or create a custom frequency when you want the schedule to run.</p> <p>Note: The "day" applies to the start time if the time range spans two days, that is, if the time range crosses the midnight boundary.</p>
Repetition	<p>If Run Once - the schedule will run once per selected day, at the time you specify.</p> <p>If Run Every - the procedure will execute repeatedly during the specified time range. For example, if you select a time range from 11:00-14:00 and an interval of 40 minutes, the procedure will execute 5 times on each of the selected days: at 11:00am, 11:40am, 12:20pm, 1:00pm, and 1:40pm.</p> <p>If Run Continuously - as soon as one job ends, the scheduler will trigger immediately to run the job again.</p>
<i>Advanced section</i>	
Enabled	If this box is "checked," the schedule will run. You can disable this schedule whenever necessary.
Misfire Policy	<p>Choose between two options to manage how a schedule resumes in cases where the normal scheduled time is interrupted. The server may not be able to trigger a job at the scheduled time for several reasons:</p> <ol style="list-style-type: none"> 1) a running job extends past the next scheduled run time, or 2) the server cannot find enough available resources and a delay occurs, or 3) the server is down or disconnected from the network for a period that overlaps one or more scheduled times. If any of these situations occur, the misfire policy defines two possible responses: <p><code>skip</code> - all misfires are ignored and the job runs at the next scheduled time.</p> <p><code>run once</code> - after one or more misfires, the job runs at the soonest time that occurs within an active region. For example, at server startup a scheduled job will not trigger immediately on startup.</p>
Time Zone	<p>Default is the time zone of the Commander server, or select a county from the first pull-down menu, then select an area from the second pull-down menu.</p> <p>Another option: Select "other" from the first pull-down menu, then select GMT [or another selection] from the second pull-down list.</p>

Field Name	Description
Priority	<p>Use the pull-down menu to select one of four priorities for this procedure.</p> <p>Also, you can select the job's priority on the Run Procedure web page. When a job is launched, its priority cannot be changed.</p> <p>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.</p> <p>If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.</p> <p>If using ectool, the priority can be set by passing <code>--priority=<low normal high highest></code> to a <code>runProcedure</code>, <code>createSchedule</code>, or <code>modifySchedule</code> operation.</p> <p>Note: Schedules are launched by the project principal user. To raise the priority above "normal," this user must have execute permission on the system-level access control list for priorities.</p>
<p><i>Impersonation Credential section</i></p> <p>If you select a credential for the schedule, it will be used as the default for jobs run from the schedule. For example, job steps will run under the login account specified in the credential. However, the schedule's credential is low in priority: it will be used only if there is no credential specified in the procedure or its project. Also, individual steps can override the credential with a specific credential for that step.</p> <p>For more information, see the Credentials and User Impersonation Help topic.</p> <p>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.</p>	
Credential Project	The name of the credential for the project. By default, the current project is selected. Supply a different project name or select Browse to find this name.
Credential Name	The name of the credential for the project - you may need to Browse to find this name.

Click **OK** to save your schedule information.

To create a new continuous integration "schedule"

If you selected the **CI Configuration** link, the New CI Configuration page is displayed. **For information** on populating these fields, see the "Continuous Integration Manager" help topic - click the **Help** link in the upper-right corner of the New CI Configuration page.

See the Standard Schedule information [above] to fill-in the fields common to both types of schedules.

To edit an existing schedule

Modify any of the information you previously specified.

The following additional links are available at the top of the page:

- **Save Configuration** - if you click this link, the information in this schedule will be copied into a new Job Configuration and displayed on your Home page for easy access.
- **Access Control** - click this link if you need to set permissions for this schedule.

Attached Credentials

Click the **Attached Credential** link if you need to attach an existing credential to this schedule. If you need to create a new credential for this schedule, return to the Project Details page.

Custom Schedule Properties

Select the **Create Properties**, **Create Nested Sheet**, or **Access Control** links if you want to assign properties to this schedule.

For more information on Properties or Access Control, see the following main Help topics: [Properties](#), [Access Control](#)

Click **OK** to save your modifications for this schedule.

Credentials - create new or modify existing credential

To create a new credential

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique name of the credential object.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
User Name	Supply the user name you choose to use for login.
Password	Supply the password that identifies the user account.

Click **OK** to save your new credential information.

To edit a credential

Modify any previously entered credential information and click **OK** to save your changes.

Use the **Access Control** link to add or change existing permissions.

You can add Custom Credential Properties for this credential also. Click the **Create Property**, **Create Nested Sheet**, or **Access Control** link in this section if you need to specify any of these items.

For more information, see the following help topics:

[Credentials and User Impersonation](#)

[Properties](#)

[Access Control](#)

Properties

- [Creating or modifying properties](#)
- [Using property values](#)
- [Property sheets and intrinsic properties](#)
- [Property names and paths](#)
- [Context-relative "shortcuts" to property paths](#)
- [Property path shortcuts in ElectricCommander 5.0 and later](#)
- [Property name substitutions](#)
- [Expandable properties](#)
- [The property hierarchy](#)
- [Special property references](#)
- [Intrinsic properties listed by object type](#)
- [Commander Object / Property tables](#)

Overview

ElectricCommander provides a powerful data model based around the notion of a *property*.

- A property is a string value with a name.
- Properties can have arbitrary names and values.
- You can attach properties to any object in the Commander system, such as a project, procedure, or job.
- After a property is created, it is stored in the Commander database.
- You can retrieve and/or modify the property value later.
- Properties you no longer need can be deleted.

Properties are used extensively throughout the Commander system and provide a flexible and powerful mechanism to manage data about your builds.

Property Use Case Examples

- When a job starts, it computes a unique identifier for that build and saves it in a property. Later build steps can retrieve the property value to embed the build number in binaries generated during the job.
- When a build executes, it can set a property on the procedure to identify the source code version used for the build (for example, a tag or timestamp from your source code control system). The next time the build executes, it can retrieve the property value from the procedure and use it to extract information from your source code control system about the files that changed since the last run.

- Suppose you have a collection of machines for testing use, and some machines have older test hardware and some machines have newer hardware with slightly different characteristics. You can set a property on each test machine resource to indicate which version of test hardware is available on which machine. Later, when a test executes, the test can retrieve the property for the machine where it ran and configure tests appropriately for that hardware. If you upgrade test hardware on a machine, all you need to do is change the property on that resource to reflect the new version.
- You can set job properties to indicate the build status produced by that job. For example, if your QA team finds fatal flaws in a build, it can mark the job accordingly. Builds that need to be preserved (release candidates or builds undergoing beta testing at customer sites) can be marked with properties so those builds are not deleted.
- When a job executes, properties are set for its job steps that hold metrics such as how many files compiled during a build step, how many tests executed during a test step, or how many errors were detected in the step. These property values are included in reports and can be examined later to compute trends over time. You can define additional properties for metrics useful to you.

Commander provides Intrinsic properties and allows you to create Custom properties. This help topic enumerates properties available within Commander (Intrinsic properties), distinguishes between relative and absolute property paths, and describes property hierarchies.

Note: Intrinsic properties are case-sensitive. Custom properties, like all other object names in the Commander system, are "case-preserving," but *not case-sensitive*.

- **Intrinsic properties**

These properties represent attributes that describe the object to which they are attached, and are provided automatically by Commander for each similar type object. For example, every project has a `Description` property that can be referenced with a non-local property path such as `/projects/Examples/description`.

- **Custom properties**

Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are *not* created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.

Creating or modifying properties

You can set a property value by using one of the following three methods or applications: the Commander web interface, the **ectool** command-line application, or the Perl API.

- Using the web interface, you will see a table labeled Custom Properties on the pages that display details for projects, procedures, and so on.
Click on the **Create New Property** action to create a new property for that object, or click on an existing property to change its value.
- Using the ectool application, set a property value with a command like:

```
ectool setProperty /myJob/installStatus complete
```

This command sets the property named `/myJob/installStatus` to the value `complete`, and creates the property if it did not already exist (the meaning of property names like

`/myJob/installStatus` is explained below). You can use `ectool` from within one job step to set properties accessed by later steps in the same job.

- Using the Perl API, you can use an external script or a script in a step with `ec-perl` as the shell. A Perl code example:

```
use ElectricCommander ();
my $ec = new ElectricCommander;
$ec->setProperty ("/myJob/installStatus", "complete", {jobStepId =>
    $ENV{COMMANDER_JOBSTEPID})
```

Note: Using the Perl API may yield better performance if you are requesting multiple API calls in one step.

For more information, review the [Using the Commander API](#) Help topic.

Using property values

A job step can access a property value by two methods. The first method is *substitution*, using the `$()` notation. Suppose you enter the following text as a command for a step:

```
make PLATFORM=$(platform)
```

Before running the step, Commander finds the property named `platform` and substitutes its value in the command string in place of `$(platform)`. For example, if the property contains the value `"windows"`, the actual command executed is:

```
make PLATFORM=windows
```

The substitution method can be used for a step's command and for any step fields, such as its resource or its working directory. This method allows you to compute configuration information in an early step of a job, then use that configuration information to control later steps in the job.

The second method for a step to access the property value is with `ectool`'s `getProperty` command. For example, the following command returns the property value named `platform`:

```
ectool getProperty platform
```

Property sheets and intrinsic properties

A property value can be a simple string or a nested *property sheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.

Most objects have an associated "property sheet" that contains "custom properties" created by user scripts. The property sheet is an intrinsic property of the containing object called "property sheet", so to reference a project's custom property `"branchName"`, you could specify `"/projects/aProject/propertySheet/branchName"`.

As a convenience, Commander allows the "property sheet" path element to be omitted and the path written as `"/projects/aProject/branchName"`. If there is no intrinsic property with the same name, the path will find the property on the property sheet.

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 via the property sheet property of their containing sheet.

For example:

```
/projects/aProject/propertySheet/topSheet/propertySheet/propB  
- or -  
/projects/aProject/topSheet/propB
```

All information managed by ElectricCommander exists in the form of properties and property sheets and your own custom-created properties. For example, each project, procedure, and step is represented internally as a property sheet—the command for a step is actually a property associated with the step, and so on. Every value in the Commander system can be accessed as a property, using the naming facilities described below. These properties are called *intrinsic* properties. **Commander enforces some restrictions on intrinsic property values, whereas custom properties can have any value you choose.**

To learn more about intrinsic properties defined for an object by ElectricCommander, consult the [ectool](#) and the [Commander API](#) help topic. For example, to learn about intrinsic properties associated with each project, find the documentation for the `getProject` ectool command. The result of running this command is an XML document whose field names and values represent the properties for the project.

Note: At the end of this help topic, find the current list of Commander intrinsic properties sorted by each object and its associated properties [in table format].

Property names and paths

Properties are named using multi-level paths such as `first/second/third`, which refers to a property named "third" in a property sheet named "second" in a property sheet named "first." ElectricCommander also supports an equivalent notation using brackets instead of slashes. For example, `first[second]/third` and `first[second][third]` both refer to the same property as `first/second/third`. The bracket `[]` notation is based on matching brackets. For example, `steps[build[1]]` refers to a property named "build[1]" inside a property sheet named "steps" (it does NOT treat "build" as a property sheet containing a property named "1").

Note: Slashes are not considered separators when they appear between brackets.

Property names/paths have two forms: *absolute* and *relative*.

Absolute property paths

Absolute property paths are referenced by a fully-qualified path syntax that begins with a slash character ("/") followed by a top-level name. This path syntax is similar to a file system path specification. The first component after the "/" must be one of several reserved words that select a starting location to look up the property name. For example, consider the property name `/server/Electric Cloud/installDirectory`. `/server` means "lookup" starts in the topmost property sheet associated with the Commander server. This property name refers to the "installDirectory" property inside the server property sheet.

The system defines the following top-level names (absolute property names):

```
/artifacts/...
```

Start in the property sheet containing all artifacts. For example, `/artifacts/myGrp:myKey/prop1` refers to the `prop1` property on the artifact whose name is "myGrp:myKey".

```
/artifactVersions/...
```

Start in the property sheet containing all artifact versions. For example,
`/artifactVersions/myGrp:myKey:1.0-36/prop1` refers to the `prop1` property on the artifact version whose name is `"myGrp:myKey:1.0-36"`. **Note:** Throughout the API, you can substitute `"groupId:artifactKey:version"` wherever an `artifactVersionName` argument can be specified. This is the same if the `artifactVersionNameTemplate` specified in the artifact is in the form `"groupId:artifactKey:version"`. For cases when the template is different, it is convenient to be able to specify this tuple if you do not know the `artifactVersionName`.

`/groups/...`

Start in the property sheet containing all groups. For example, `/groups[dev]` refers to the group named `"dev"`.

`/projects/...`

Start in the property sheet containing all projects. For example, `/projects[nightly]` refers to the project named `"nightly,"` and `/projects[nightly]/procedures[main]` refers to a procedure named `"main"` in that project.

`/jobs/...`

Start in the property sheet containing all jobs. For example, `/jobs[ecloud.4096]` refers to the job named `"ecloud.4096."` You can name a job using either its name (as in the preceding example) or using the unique identifier assigned to it by Commander.

`/plugins/...`

Start in the property sheet containing all plugins. For example, `/plugins[EC-AgentManagement]` refers to the currently promoted plugin named `"EC-AgentManagement"` and `/plugins[EC-AgentManagement]/project/procedures[scpCopyFile]` refers to a procedure named `"scpCopyFile"` in that plugin.

Note: There is a subtle difference between

```
ectool setProperty /plugins/EC-AgentManagement/project/foo 'bar'
```

and

```
ectool setProperty /plugins/EC-AgentManagement/foo 'bar'
```

The former places the property on the plugin's project and can be referenced by `$/myProject/foo` while the latter places the property on the plugin object and will not be found via `$/myPlugin/foo`.

`/repositories/...`

Start in the property sheet containing all artifact repositories. For example,
`/repositories/repo1/prop1` refers to the `prop1` property on the repository whose name is `"repo1"`.

`/resourcePools/...`

Start in the property sheet containing all resource pools. For example, `/resourcePools[linuxA]` refers to the resource pool named `"linuxA"`.

`/resources/...`

Start in the property sheet containing all resources. For example, `/resources[linux1]` refers to the resource named "linux1".

`/server/...`

Start in the top-level server property sheet. For example, `/server/a` refers to the server property named "a".

`/users/...`

Start in the property sheet containing all users. For example, `/users[Bob]` refers to the user named "Bob".

`/workspaces/...`

Start in the property sheet containing all workspaces. For example, `/workspaces[default]` refers to the workspace named "default."

Relative property paths

Property names that do not begin with "/" are *relative* and looked up starting in the *current context*. For example, when executing a job step, the current context includes properties defined for the job step, parameters for the current procedure, and global properties on the current job.

Relative property paths are distinguished from absolute property paths because they do not begin with one of the top-level names. To avoid having to construct full property paths, Commander supports the concept of a relative property path.

In use, the relative property path value is resolved by its context and a defined search order, which results in accessing the value of an absolute fully-specified property value. Contexts and search orders are as follows:

1. In a job step context, Commander searches for relative property paths in the following order:
 - a property on the job step object
 - a property of the parent job step object, which includes parameters of the procedure on which the step is defined
 - a property on the job object

Notes:

- The search can be enabled or disabled by using the `--extendedContextSearch` option on the `ectool getProperty` or `setProperty` commands. When searching for a property value, disable the search by setting the `--extendedContextSearch` switch to "false", requiring the property to exist on the job step object or return false. When writing a new value to a property, enable the search by setting the `--extendedContextSearch` switch to "true", allowing the search for a property within the search order before creating a new one. New properties are created on the job step object.
 - Parameters for subprocedure calls from job steps are searched for in a job step context.
 - For procedure parameters for nested subprocedures, properties referenced by parameters are looked up as described [above] for job steps.
2. When expanding schedule parameters, Commander searches the relative property path in the following order:

- a property on the procedure being called
 - a property on the project on which the procedure being called is defined
 - a property on the server
3. In a job name template context, Commander searches for the relative property path as a property on the job.
 4. In any other context, Commander searches for the relative property path as a property on the context object.

Context-relative "shortcuts" to property paths

A 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. 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.

Shortcuts to property paths that provide convenient runtime access include:

`/myArtifactVersion/...`

Start in the property sheet for the `artifactVersion` associated with the current context. The only context where this property has any value is in the `artifactVersionNameTemplate` field for the artifact.

`/myCall/...`

`/myCall/` is now a subset of `/myParent/`. (See the `/myParent/` description below.) `/myCall/` refers to the parent (job or jobStep's) CUSTOM property sheet, so you can reference custom properties and parameters only, which get copied into the custom property sheet. `/myJob/`, on the other hand, points directly to the job object, so you can reference intrinsic job properties in addition to the properties in the custom property sheet.

`/myCredential/...`

Start in the property sheet for the credential associated with the current job step. This form produces an error if the current job step does not have a credential.

`/myEvent/...`

This is a special property used only within an "email notifier." `/myEvent/` allows you to refer to fields associated with the notifier itself. You can include these properties, for example, in the text of the email you send out for notification:

`/myEvent/notifier` - This property contains the name of the notifier that created the notifier event. You created this name when you created the notifier.

`/myEvent/entity` - This property contains the object where the notifier is attached. Two possible values are "job" or "jobStep," depending on where the notifier is attached.

`/myEvent/source` - This property contains the name of either the job or the jobStep where the notifier is attached.

`/myEvent/type` - This property defines whether the notifier is an "On Start" or an "On Completion" notifier. Two possible values are "STARTED" or "COMPLETED".

`/myEvent/time` - This property contains the time when the notifier occurred. The time is always specified in GMT and uses a formatted string, for example,
2009-06-11T21:00:56.502Z

`/myEvent/timeMillis` - This property contains the "timestamp" in milliseconds when the notifier occurred. This value is the number of milliseconds since January 1, 1970 GMT. The `timeMillis` corresponding to the time in the previous example is: 1244754056502

`/myJob/...`

Start in the global property sheet for the current job. `/myJob/` points directly to the job object so you can reference built-in job properties (`jobName`, `createTime`, `outcome`, and so on) as well as custom properties. Also, this property sheet can be used to hold working data that needs to be passed from one step to another within the job.

`/myJobStep/...`

Start in the property sheet for the current job step.

`/myParent/...`

Start in the global property sheet for the parent job step or job if this is a top-level step. `/myParent/` points directly to the job or job step object, so you can reference intrinsic or custom properties. For example, you can reference the parameters that were passed to this procedure.

In addition, you can reference a sibling step by calling `/myParent/jobSteps/<sibling step name>`.

Or, you can create additional properties on `/myParent/` to pass information from one step in the procedure to another:

step1 calls `setProperty /myParent/results 100`

step2 can call `getProperty /myParent/results`

`/myParent/` is a superset of `/myCall/`. You can continue to use `/myCall/`, but Electric Cloud recommends using `/myParent/` in new scripts you create.

`/myProcedure/...`

Start in the property sheet for the procedure in which the current job step was defined. If the current job step is executing as part of a nested procedure, `/myProcedure/` refers to the innermost nested procedure.

`/myProject/...`

Start in the property sheet for the project in which the current job step was defined. In the case of nested procedure invocations, this is the project associated with the innermost nested procedure, for example, the project associated with `/myProcedure/`.

`/myResource/...`

Start in the property sheet for the resource assigned to the current job step.

`/myResourcePool/...`

Start in the property sheet for the resource pool that provided the resource for the current job step. Returns null if the step did not specify a resource pool.

`/myStep/...`

Start in the property sheet for the current step. "Step" refers to the (static) definition of a step, which is part of a procedure— this is different from a job step, which represents a step when it executes (dynamically) in a job. Use `/myJobStep/` to access the job step.

`/myState/...`

Start in the property sheet for the state object so you can reference built-in and custom state properties or find parameter values passed to that state. When accessed from a state, `/myState/` refers to that state. When accessed from a transition, `/myState` refers to the transition's owning state. When accessed from a job or job step, `/myState/` refers to the state that launched that job as a subjob.

`/mySubjob/...`

Start in the property sheet for the subjob so you can reference built-in and custom job properties, parameters passed to the job, and properties on steps within that job. When accessed from a transition, `/mySubjob/` refers to the subjob started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the outcome or other information for the subjob can influence which state the workflow transitions to next.

`/mySubworkflow/...`

Start in the property sheet for the subworkflow so you can reference built-in and custom workflow properties. When accessed from a transition, `/mySubworkflow/` refers to the subworkflow started by the state that owns that transition. This property path is particularly useful in conditions for On Completion transitions because the active state and other information for the subworkflow can influence which state the workflow transitions to next. You can access information about states and transitions belonging to the workflow by using the path `/mySubworkflow/states/someState` or `/mySubworkflow/states/someState/transitions/someTransition`.

`/myTransition/...`

Start in the property sheet for the transition object so you can reference intrinsic and custom transition properties. `/myTransition/` is accessible only from a transition.

`/myUser/...`

This property can be used only if the current session is associated with: the predefined "admin" user, a user defined as "local", or a user defined by a Directory Provider (LDAP or ActiveDirectory). This property cannot be used if the user is a "project principal", which is normally the case when running inside a Commander step.

For example, with an interactive login you can use:

```
ectool getProperty /myUser/userName (to get the user name of the logged in user, or...)
ectool getProperty /myUser/email (to get the email address)
```

`/myWorkflow/...`

Start in the property sheet for the workflow object so you can reference built-in and custom workflow properties. When accessed from a state or transition, `/myWorkflow/` refers to the "owning" workflow. When accessed from a job or job step, `/myWorkflow/` refers to the workflow whose state launched that job as a subjob. You can access information about states and transitions belonging to the workflow by using the path `/myWorkflow/states/someState` or `/myWorkflow/states/someState/transitions/someTransition`.

`/myWorkspace/...`

Start in the property sheet for the workspace associated with the current job step.

Property path shortcuts in ElectricCommander 5.0 and later

The following shortcuts to property paths are available in ElectricCommander and later. For more information, go to [Context-relative "shortcuts" to property paths](#).

Shortcuts	Descriptions	For jobs	For job steps
<code>/myApplication/...</code>	Starts in the property sheet for the application associated with the current job or job step.	Yes	Yes
<code>/myApplicationTier/...</code>	Starts in the property sheet for the application tier associated with the current job step.	No	Yes
<code>/myCall/...</code>	Starts in the property sheet for the call associated with the current job step.	No	Yes
<code>/myComponent/...</code>	Starts with the property sheet for the component associated with the current job step.	No	Yes
<code>/myCredential/...</code>	Starts with the property sheet for the credential associated with the current job step.	No	Yes
<code>/myEnvironment/...</code>	Starts in the property sheet for the environment associated with the current job or job step.	Yes	Yes
<code>/myEnvironmentTier/...</code>	Starts in the property sheet for the environment tier associated with the current job step.	No	Yes
<code>/myJob/...</code>	Starts in the property sheet for the job associated with the current job or job step.	Yes	Yes
<code>/myJobStep/...</code>	Starts in the property sheet for the job step associated with the current job step.	No	Yes

Shortcuts	Descriptions	For jobs	For job steps
/myParent/...	Starts in the global property sheet for the parent job step.	No	Yes
/myPlugin/...	Starts in the property sheet for the plugin associated with the current job.	Yes	No
/myProcedure/...	Starts in the property sheet for the procedure in which the current job or job step was defined.	Yes	Yes
/myProcess/...	Starts in the property sheet for the process in which the current job or job step was defined.	Yes	Yes
/myProcessStep/...	Starts in the property sheet for the process step in which the current job step was defined.	No	Yes
/myProject/...	Starts in the property sheet for the project in which the current job was defined.	Yes	No
/myResource/...	Starts in the property sheet for the resource associated with the current job step.	No	Yes
/myResourcePool/...	Starts in the property sheet for the resource pool associated with the current job step.	No	Yes
/myState/...	Starts in the property sheet for the state object so you can reference built-in and custom state properties or find parameter values passed to that state.	Yes	Yes
/myStep/...	Starts in the property sheet for the step associated with the current job step.	No	Yes
/myWorkflow/...	Starts in the property sheet for the workflow object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkflowDefinition/...	Starts in the property sheet for the workflow definition object so you can reference built-in and custom workflow properties.	Yes	Yes
/myWorkspace/...	Starts in the property sheet for the workspace associated with the current job step.	No	Yes

Property name substitutions

Property names can contain references to other properties, which are then substituted into the property name before looking it up. For example, consider the following property name:

```
/myStep/${/myProcedure/name}
```

If the value of `/myProcedure/name` is "xyz", the property above is equivalent to `/myStep/xyz`.

Expandable properties

Property values can contain property references using the `"${}"` notation.

For example:

1. Create a property named "foo" with a value of `hello ${bar}`.
2. Create a property named "bar" with a value of `world`.
3. Reference "foo" (either using `${foo}` or `ectool getProperty foo`). The value "hello world" is returned.

If you want just the literal value of "foo" (useful in the UI, for example), you can use the `expand` option in `ectool`:

```
ectool getProperty foo --expand false. The value "hello ${bar}" will be returned.
```

Properties are expanded by default when you use `getProperty` or `getProperties`.

If the value of a property contains `"${}"` but you do not want it to be interpreted as a property reference, you can use the `expandable` option:

```
ectool setProperty symbols '${!@}#' --expandable false.
```

This option can be toggled in the web UI as well. Properties are expandable by default.

Because you cannot control where your expandable property might be referenced (and therefore which context is used during expansion), Electric Cloud recommends using absolute paths when referencing a property from the value of another property.

In the example above, if you define both "foo" and "bar" as properties on a project "proj1", you might assume there is no problem with the value of "foo". However, if you later reference "foo" from a job under "proj1" (for example, `${/myProject/foo}`), `foo` will be referenced with the job step as its context. Therefore, when the value of "foo" is expanded, you will get a `PROPERTY_REFERENCE_ERROR` because "bar" is not defined in the context of the job step.

Custom property names and values

The following properties are used by the standard ElectricCommander UI, so you should use these property names whenever possible, and avoid using these names in ways that conflict with the definitions below.

- **preSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *before* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

- **postSummary** - if this property exists, its value is displayed in the "Status" field (on the Job Details page) for this step. This property appears *after* whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.
- **summary** - if this property exists, its value is displayed in the "Status" field (in the job reports) for this step, replacing whatever would normally be displayed for status. If the property contains multiple lines separated by newline characters, each line is displayed on a separate line in the status field.

The property hierarchy

All ElectricCommander properties fall into a hierarchical structure, and you can reference any property using an absolute path from the root of the hierarchy. This includes properties you define and intrinsic properties defined by Commander.

For example, each step contains a property "resource" that provides the resource name to use for that step. All steps of a procedure exist as property sheets underneath the procedure. All procedures in a project exist as property sheets underneath the project, and so on.

The examples below illustrate some nesting relationships between objects.

For example, the notation "*project/procedures[procedureName]*" means each project object contains a property sheet named "procedures" holding all procedures defined within that project. Within the procedures property sheet, there is a nested property sheet for each procedure, named after the procedure. Thus, the name "*/projects[a]/procedures[b]*" refers to a procedure named "b" contained in a project named "a."

- Each project contains procedures, schedules, credential definitions, workflow definitions, and workflows:

```
project/procedures[procedureName]
project/schedules[scheduleName]
project/credentials[credentialName]
project/workflowDefinitions[workflowName]
project/workflows[workflowName]
```

- Each procedure contains steps and parameters. The parameters are "formal parameters," meaning they specify parameter names the procedure will accept, whether each parameter is required, and so on:

```
procedure/steps[stepName]
procedure/formalParameters[parameterName]
```

- Steps that invoke subprocedures contain parameter values for the subprocedure. These are called "actual parameters" because they provide actual values that will be passed into the subprocedure:

```
step/actualParameters[parameterName]
```

- Each job contains a collection of job step objects for steps in the outermost procedure, along with parameter values passed into the job when it was invoked:

```
job/jobSteps[stepName]
job/actualParameters[parameterName]
```

- If a job step invokes a nested subprocedure, its property sheet contains parameter values that were passed into the nested subprocedure, plus all job steps corresponding to that procedure:

```
jobStep/actualParameters[parameterName]
jobStep/jobSteps[stepName]
```

- Schedules contain parameter values for the procedures they invoke. These are called "actual parameters" because they provide actual values passed into the procedure:
`schedule/actualParameters[parameterName]`
- Workflow definitions contain state definitions:
`workflowDefinition/stateDefinitions[stateDefinitionName]`
- Transition definitions contain actual parameters:
`transitionDefinition/actualParameters[parameterName]`
- State definitions contain transition definitions, actual parameters, and formal parameters:
`stateDefinition/actualParameters[parameterName]`
`stateDefinition/formalParameters[parameterName]`
`stateDefinition/transitionDefinitions[transitionDefinitionName]`
- Workflows contains states:
`workflow/states[stateName]`
- Transitions contain actual parameters:
`transition/actualParameters[parameterName]`
- States contain transitions, actual parameters, and formal parameters:
`state/actualParameters[parameterName]`
`state/formalParameters[parameterName]`
`state/transitions[transitionName]`

Special property references

ElectricCommander also supports several special property reference forms that are described in the following subsections.

increment

Use this form to increment the value of an integer property before returning its value. For example, suppose property xyz has the value 43. The property reference `$[/increment xyz]` first increments the value of property xyz to 44, then returns 44.

timestamp

Use this form to generate a formatted timestamp value. For example, the property reference `$[/timestamp yyyy-MM-d hh:mm]` returns the current time in a form such as "2007-Jun-19 04:36". The pattern following `/timestamp` specifies how to format the time and defaults to "yyyyMMddHHmm". The pattern follows conventions for the Java class `SimpleDateFormat`, where various letters are substituted with various current time elements. For more information, go to ["http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html"](http://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html) for `SimpleDateFormat` class documentation.

Here are some of the supported substitutions:

y - Year, such as "2007" or "07"

M - Month, such as "April", "Apr", or "04"

w - Week in year, such as "27"

W - Week in month, such as "3"

D - Day in year, such as "194"

d - Day in month, such as "18"

E - Day in the week, such as "Tuesday" or "Tue"

a - am/pm marker, such as "PM"

H - Hour in day (0-23), such as "7"

h - Hour in am/pm (1-12), such as "11"

m - Minute in hour, such as "30"

s - Second in minute, such as "15"

S - Millisecond, such as "908"

z - Time zone such as "Pacific Standard Time", "PST", or "GMT-08:00"

Z - Time zone such as "-0800"

Repeated letters cause longer forms to be substituted. For example, "yy" substitutes the last 2 digits of the current year, whereas "yyyy" substitutes the 4-digit year number. Single quotes can be used to substitute text directly without the above interpretations.

javascript

This form executes a Javascript code fragment inside the Commander server and returns the result computed by that code.

For example, `$/javascript 4*2` returns "8.0". Javascript code can be arbitrarily long and include multiple statements. The value of the last statement is returned by the property reference.

Javascript code executes in an interpreter that provides access to Commander properties:

1. The normal way to access property values is through Javascript objects. Global Javascript objects named "server," "projects," and so on, exist and correspond to all top-level objects in an absolute property path.
For example, there is a Javascript object named `server` that corresponds to the path `/server`, and a Javascript object named `myJob` that corresponds to the path `/myJob`. You can use either `.` or `[]` notation to access properties within the object.
For a more complete list of top-level objects, see the ["Absolute property paths"](#) section above.

Examples:

```
$/javascript myJob.mightExist]
this is a safe way to refer to an optional parameter
```

```
$/javascript (myJobStep.errors > 0) ? "step failed" : "no errors"]
test a value and return another string based on the result
```

```
$/javascript server.settings.ipAddress]
refer to a property in a nested property sheet
```

```
$/javascript server["Electric Cloud"].installDirectory]
```

if the property name has a space, use `[" "]` notation

```
$[/javascript server["Electric Cloud"]["installDirectory"]]
```

do not use the `."` in front of the `["`

2. You can also call the function, `getProperty`. It takes a property name as argument and returns the value of that property. The case where this is most useful is when you want to access another special property reference.

Examples:

```
$[/javascript getProperty ("/timestamp yyyy-MMM-d hh:mm")]
```

returns the current time

```
$[/javascript getProperty ("/increment /myJob/jsCount")]
```

this is the only way to change a property via javascript

For example, consider the following property reference:

```
$[/javascript (getProperty("/myJobStep/errors") > 0) ? "step failed" : "no errors"]
```

This example returns "step failed" if the property "errors" on the current job step had a value greater than zero, and it returns "no errors" otherwise.

3. If calling the `setProperty` function, similar to `getProperty`, there are two variations on the function:
 - A *global* function variation that uses the current context object.
The global function takes 2 or 3 arguments. The 3-argument version takes a context object, a path, and a value. The 2-argument version omits the context object and uses the current context object (for example, *job step*).

Example:

```
setProperty (myProject, "foo", "bar")
```

would set the value of the "foo" property on the current project to "bar".

- An *object* function variation that can be called on objects, which uses that object as a context object. The object function takes two arguments: a "path" and a "value".

Example:

```
server.setProperty ("foo", "bar")
```

would set the "foo" property on the server object to the value "bar".

Commander Intrinsic Properties listed by object type

Each of the Commander object types [in the list below] links to a table for that object.

Each table includes a list of intrinsic properties applicable to that object and provides the property type and

description.

acl	repository
artifact	resource
artifactVersion	resourcePool
credential	resourceUsage
directoryProvider	schedule
emailConfig	server
emailNotifier	state
formalParameter	stateDefinition
gateways	step
group	transition
job	transitionDefinition
jobStep	user
logEntry	workflow
procedure	workflowDefinition
project	workspace
property	zones
propertySheet	

Commander Property Type definitions

This table provides Property Type definitions for each property listed in the following tables.
(A Property Type precedes each property description in the Description column.)

Type	Definition
boolean	One of two possible values - either <i>true</i> or <i>false</i> . These values are more frequently represented by the numbers "0" and "1", where "0" equals false and "1" equals true.
date	A millisecond precision UTC date in ISO 8601 form: [YYYY] - [MM] - [DD] T [hh] : [mm] Z For example, 2007-06-19T04:36:22.000Z
id	Each time an object is created, ElectricCommander generates a unique ID number for that object.
name	This is a unicode string value with a maximum of 255 characters.
number	This is a simple integer numeric value.
reference	This property refers to another object.
string	This is a unicode string value with a maximum CLOB size of the database, but only the first 450 characters are indexed, which means a defined search will not "see" beyond the first 450 characters.

Commander Object / Property tables

In the following tables, the Description column displays the property type preceding the property description.

Object Type: <code>acl</code> Description: An <i>acl</i> is an Access Control List.	
Property Name	Description
<code>aclId</code>	<code>id</code> : The unique identifier for this <code>acl</code> object. Other objects can refer to this <code>acl</code> by its ID.
<code>inheriting</code>	<code>boolean</code> : If true, the ACL inherits ACEs from the ACL's parent.

Object Type: <code>acl</code> Description: An <i>acl</i> is an Access Control List.																													
Property Name	Description																												
<code>ownerType</code>	<p>This is any type of object the ACL controls and can be any of the objects listed below.</p> <table> <tr> <td><code>ace</code></td><td><code>project</code></td></tr> <tr> <td><code>acl</code></td><td><code>property</code></td></tr> <tr> <td><code>admin</code></td><td><code>propertySheet</code></td></tr> <tr> <td><code>artifact</code></td><td><code>repository</code></td></tr> <tr> <td><code>artifactVersion</code></td><td><code>resource</code></td></tr> <tr> <td><code>event</code></td><td><code>resourcePool</code></td></tr> <tr> <td><code>formalParameter</code></td><td><code>server</code></td></tr> <tr> <td><code>group</code></td><td><code>schedule</code></td></tr> <tr> <td><code>job</code></td><td><code>systemObject</code></td></tr> <tr> <td><code>jobStep</code></td><td><code>user</code></td></tr> <tr> <td><code>license</code></td><td><code>userSettings</code></td></tr> <tr> <td><code>notifier</code></td><td><code>workspace</code></td></tr> <tr> <td><code>procedure</code></td><td><code>plugin</code></td></tr> <tr> <td><code>procedureStep</code></td><td></td></tr> </table>	<code>ace</code>	<code>project</code>	<code>acl</code>	<code>property</code>	<code>admin</code>	<code>propertySheet</code>	<code>artifact</code>	<code>repository</code>	<code>artifactVersion</code>	<code>resource</code>	<code>event</code>	<code>resourcePool</code>	<code>formalParameter</code>	<code>server</code>	<code>group</code>	<code>schedule</code>	<code>job</code>	<code>systemObject</code>	<code>jobStep</code>	<code>user</code>	<code>license</code>	<code>userSettings</code>	<code>notifier</code>	<code>workspace</code>	<code>procedure</code>	<code>plugin</code>	<code>procedureStep</code>	
<code>ace</code>	<code>project</code>																												
<code>acl</code>	<code>property</code>																												
<code>admin</code>	<code>propertySheet</code>																												
<code>artifact</code>	<code>repository</code>																												
<code>artifactVersion</code>	<code>resource</code>																												
<code>event</code>	<code>resourcePool</code>																												
<code>formalParameter</code>	<code>server</code>																												
<code>group</code>	<code>schedule</code>																												
<code>job</code>	<code>systemObject</code>																												
<code>jobStep</code>	<code>user</code>																												
<code>license</code>	<code>userSettings</code>																												
<code>notifier</code>	<code>workspace</code>																												
<code>procedure</code>	<code>plugin</code>																												
<code>procedureStep</code>																													
<code>parentId</code>	<code>id</code> : The parent ACL.																												

[\[back to top\]](#)

Object Type: <code>artifact</code> Description: An <i>artifact</i> is an object that contains zero or more artifact versions. An artifact has two purposes: <ol style="list-style-type: none"> 1. To group artifact versions and provide a template for naming the versions 2. To restrict who can publish artifact versions, based on <code>groupId:artifactKey</code> 	
Property Name	Description
<code>acl</code>	<code>reference:acl</code>
<code>artifactId</code>	<code>id</code> : The artifact's ID number.

Object Type: artifact

Description: An *artifact* is an object that contains zero or more artifact versions.

An artifact has two purposes:

1. To group artifact versions and provide a template for naming the versions
2. To restrict who can publish artifact versions, based on `groupId:artifactKey`

Property Name	Description
<code>artifactKey</code>	<code>string</code> : User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>artifactName</code>	<code>name</code> : The name of this artifact.
<code>artifactVersionNameTemplate</code>	<code>name</code> : The template for artifact version names published to this artifact.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>groupId</code>	<code>id</code> : A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>

[\[back to top\]](#)

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
acl	<code>reference:</code> <code>acl</code>
artifactKey	<code>string:</code> User-specified identifier for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName	<code>name:</code> The name of the artifact.
artifactVersionId	<code>id:</code> The Commander-generated ID number for this artifact version.
artifactVersionName	<code>name:</code> The name of the artifact version.
artifactVersionState	<code>string:</code> Possible values are: <code>available publishing unavailable</code>
buildNumber	<code>number:</code> User-defined build number component of the version attribute for the artifact version.
createTime	<code>date:</code> The time when this object was created.
description	<code>string:</code> A user-specified text description of the object.
groupId	<code>id:</code> A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.
lastModifiedBy	<code>name:</code> This shows who (generally, a user name) last modified this object.
majorMinorPatch	<code>string:</code> <code>major.minor.patch</code> component of the version attribute for the artifact.
modifyTime	<code>date:</code> The time when this object was last modified.
owner	<code>name:</code> The person (user name) who created the object.

Object Type: artifactVersion

Description: An artifact version is an object that represents a user-defined unit of related files typically produced by one job and consumed by one or more other jobs.

Property Name	Description
propertySheet	<code>reference</code> : <code>propertySheet</code>
publisherJobId	<code>id</code> : The Commander-generated ID number for the job that published the artifact version.
publisherJobName	<code>string</code> : The name of the job that published the artifact version.
publisherJobStepId	<code>id</code> : The Commander-generated ID number for the job step that published the artifact version.
qualifier	<code>string</code> : User-defined qualifier component of the version attribute for the artifact.
repositoryName	<code>name</code> : The name of the artifact repository.
version	<code>string</code> : An artifact version specification uses the following form: <i>major.minor.patch.qualifier.buildNumber</i>

[\[back to top\]](#)

Object Type: credential

Description: In Commander, a *credential* is an object that stores a username and password for later use.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
createTime	<code>date</code> : The time when this object was created.
credentialId	<code>id</code> : The credential's ID number.
credentialName	<code>name</code> : The name of this credential.

Object Type: credential

Description: In Commander, a *credential* is an object that stores a username and password for later use.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
password	<code>string</code> : The password corresponding to the user name and this credential.
projectName	<code>name</code> : The name of the <code>project</code> that contains this credential.
propertySheet	<code>reference</code> : <code>propertySheet</code>
userName	<code>name</code> : A saved string that represents the name portion of a credential, typically a user account name.

[\[back to top\]](#)

Object Type: directoryProvider

Description: A `directoryProvider` is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
commonGroupNameAttribute	<code>string</code> : 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.

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
directoryProviderId	<code>id</code> : The ID of the directory provider.
domainName	<code>string</code> : The domain name from which Active Directory server(s) are automatically discovered.
emailAttribute	<code>string</code> : The attribute in a user record that contains the user's email address. If the attribute was not specified, the account name and domain name are concatenated to form an email address.
enableGroups	<code>boolean</code> : Determines whether or not external groups are enabled for the directory provider. Defaults to "true".
fullUserNameAttribute	<code>name</code> : The attribute in a user record that contains the user's full name (first and last) for display in the UI.
groupBase	<code>string</code> : This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
groupMemberAttributes	<code>string</code> : A comma-separated attribute name list that identifies a group member.
groupMemberFilter	<code>string</code> : Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	<code>name</code> : The group record that contains the name of the group.

Object Type: `directoryProvider`

Description: A `directoryProvider` is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
<code>groupSearchFilter</code>	<code>string</code> : This LDAP query is performed in the context of the groups directory to enumerate group records.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>managerDn</code>	<code>name</code> : The DN of a user who has read-only access to LDAP user and group directories.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>providerIndex</code>	<code>number</code> : The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index "0" and one with index "1" means the providers will be searched in that numerical order.
<code>providerName</code>	<code>name</code> : This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.
<code>providerType</code>	<code>string</code> : <code><ldap activedirectory></code>
<code>realm</code>	<code>string</code> : An identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <code><user>@dir</code> (where the realm is set to "dir").
<code>url</code>	<code>string</code> : The server URL is in the form <code>protocol://host:port/basedn</code> . Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP).

Object Type: directoryProvider

Description: A directoryProvider is an object containing information about an external directory service (LDAP or ActiveDirectory).

Property Name	Description
userBase	<code>string</code> : This string is prepended to the <code>basedn</code> to construct the directory DN that contains user records.
userNameAttribute	<code>name</code> : The attribute in a user record that contains the user's account name.
userSearchFilter	<code>string</code> : This LDAP query is performed in the context of the user directory to search for a user by account name. The string "{0}" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
userSearchSubtree	<code>boolean</code> : If true, the subtree below the user base is searched recursively.
useSSL	<code>boolean</code> : This flag is used to specify SSL to communicate with your Active Directory servers.

[\[back to top\]](#)**Object Type: emailConfig**

Description: An *emailConfig* is an object that stores information created and used to communicate with the email server.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
configName	<code>string</code> : The name of the email configuration.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.

Object Type: emailConfig Description: An <i>emailConfig</i> is an object that stores information created and used to communicate with the email server.	
Property Name	Description
emailConfigId	<code>id</code> : The Commander-generated ID for the email configuration.
emailConfigName	<code>string</code> : The name of the email configuration.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
mailFrom	<code>string</code> : The email address used as the email sender address for notifications.
mailHost	<code>string</code> : The name of the email server host.
mailPort	<code>number</code> : The port number for the mail server. The protocol software determines the default value (25 for SMTP and 465 for SSMTP).
mailProtocol	<code>string</code> : This is either SSMTP or SMTP (not case sensitive). Default is SMTP.
mailUser	<code>name</code> : An individual or a generic name like "Commander" -- the name of the email user on whose behalf Commander sends email notifications.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : propertySheet

[\[back to top\]](#)

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
acl	<code>reference:</code> <code>acl</code>
condition	<code>string</code> : Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
configName	<code>string</code> : The name of the email configuration.
container	<code>id</code> : The object ID to which the email notifier is attached (for example: procedure-123).
containerType	<code>string</code> : The type of object that the notifier is attached to (procedure, step, job, jobstep).
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
destinations	<code>string</code> : A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.
emailNotifierId	<code>id</code> : The Commander-generated ID for the email notifier.
eventType	<code>string</code> : <code><onEnter onStart onCompletion></code> <code>onStart</code> triggers an email notification when the job or job step begins. <code>onCompletion</code> , default, triggers an email notification when the job finishes, no matter how it finishes.
formattingTemplate	<code>string</code> : The email address used as the email sender address for notifications.

Object Type: emailNotifier

Description: An *emailNotifier* is an object that stores information created and used to notify users about various types information, including status.

Property Name	Description
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
notifierName	<code>name</code> : The name of the email notifier.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : propertySheet

[\[back to top\]](#)

Object Type: formalParameter

Description: A *formalParameter* is 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 type of parameters a procedure is expecting, and actual parameters provide values to use at run-time.

Property Name	Description
container	<code>string</code> : An object ID for a "container" that contains formal parameters.
containerType	<code>string</code> : The type of object containing the formal parameter.
createTime	<code>date</code> : The time when this object was created.
defaultValue	<code>string</code> : The <code>formalParameter</code> 's default value.

Object Type: `formalParameter`

Description: A *formalParameter* is 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 type of parameters a procedure is expecting, and actual parameters provide values to use at run-time.

Property Name	Description
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>expansionDeferred</code>	<code>boolean</code> : Determines whether or not the property expansion is deferred.
<code>formalParameterId</code>	<code>id</code> : This is this formal parameter's ID.
<code>formalParameterName</code>	<code>name</code> : This is this formal parameter's name.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>required</code>	<code>boolean</code> : If true, a value for this parameter must be supplied when the procedure is called.
<code>type</code>	<code>name</code> : The custom type of <code>formalParameter</code> .

[\[back to top\]](#)

Object Type: gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
acl	<code>reference</code> : acl
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
gatewayDisabled	<code>boolean</code> : If set to 1 (true), the gateway is disabled.
gatewayId	<code>id</code> : The unique Commander-generated ID for this gateway.
gatewayName	<code>string</code> : The name of the gateway.
hostName1	<code>string</code> : The agent host name where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option if you want to use the host name from <i>Resource1</i> 's definition.
hostName2	<code>string</code> : The agent host name where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option if you want to use the host name from <i>Resource2</i> 's definition.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
port1	<code>number</code> : The port number used by <i>Resource1</i> - defaults to the port number used by the resource.

Object Type: gateway

Description: To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

Property Name	Description
port2	<code>number</code> : The port number used by <i>Resource2</i> - defaults to the port number used by the resource.
propertySheet	<code>reference</code> : <code>propertySheet</code>
resourceName1	<code>string</code> : The name of your choice for the first of two required gateway resources. Do not include "spaces" in a resource name.
resourceName2	<code>string</code> : The name of your choice for the second of two required gateway resources. Do not include "spaces" in a resource name.

[\[back to top\]](#)**Object Type: group**

Description: This is any *group* of users known to the Commander server, including groups defined locally within the server as well as those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
createTime	<code>date</code> : The time when this object was created.
groupId	<code>id</code> : The unique Commander-generated group ID.
groupName	<code>name</code> : This is this group's name.

Object Type: group

Description: This is any *group* of users known to the Commander server, including groups defined locally within the server as well as those groups defined in external repositories such as LDAP or ActiveDirectory.

Property Name	Description
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>mutable</code>	<code>boolean</code> : If true, the member list of this group is editable within ElectricCommander via the web UI or the <code>modifyGroup</code> API.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>providerName</code>	<code>name</code> : The name of the <code>directory provider</code> that controls this group.

[\[back to top\]](#)

Object Type: job

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>abortStatus</code>	<code>string</code> : If set, indicates the step was aborted. Values will be either <code>ABORT</code> or <code>FORCE_ABORT</code> - indicating how the step was aborted.
<code>abortedBy</code>	<code>name</code> : This is the user who issued the "abort."

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>actualParameters</code>	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>callingState</code>	<code>string</code> : The full property path to the "calling state", which can appear on <code>subjobs</code> and <code>subworkflows</code> of a workflow.
<code>callingStateId</code>	<code>id</code> : The Commander-generated ID number for the calling state.
<code>combinedStatus</code>	Provides more inclusive step status output - the resulting query output may contain up to three sub-elements: <code>status message properties</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>credentialName</code>	<code>name</code> : The name of the <code>credential</code> being used for impersonation when the job runs commands on a resource.
<code>deleted</code>	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
<code>directoryName</code>	<code>name</code> : The name of this job's directory within each workspace for this job.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>elapsedTime</code>	<code>number</code> : The number of milliseconds between the start and end times for the job.
<code>errorCode</code>	<code>errorCode</code> : When the outcome is <code>error</code> , this property displays the error code, identifying which error occurred.
<code>errorMessage</code>	<code>string</code> : When the outcome is <code>error</code> , this property displays the error description.
<code>external</code>	<code>boolean</code> : If "true", the job is external.
<code>finish</code>	<code>date</code> : The time this job completed.
<code>jobId</code>	<code>id</code> : This is this job's ID number, which is a UUID.
<code>jobName</code>	<code>name</code> : This is this job's name.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>launchedByUser</code>	<code>name</code> : 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.
<code>licenseWaitTime</code>	The sum of time all job steps had to wait for a license.
<code>liveProcedure</code>	<code>name</code> : The current procedure name from which this job was created – if the procedure was renamed since the job launched, this will be the procedure's new name, and if the procedure was deleted, this will be null.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>liveSchedule</code>	<code>name</code> : The current schedule name that launched this job – if the schedule was renamed since the job was launched, this will be the schedule's new name, and if the schedule was deleted, this will be null.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>outcome</code>	The overall result of the job: <code>success</code> , <code>warning</code> , <code>error</code> , or <code>skipped</code> .
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>priority</code>	Values can be <code>low</code> , <code>normal</code> (default), <code>high</code> , or <code>highest</code> .
<code>procedureName</code>	<code>name</code> : The name of the <code>procedure</code> that defines the job's steps.
<code>projectName</code>	<code>name</code> : The name of the <code>project</code> that contains this job.
<code>propertySheetId</code>	<code>reference</code> : <code>propertySheet</code>
<code>resourceWaitTime</code>	The sum of time all job steps had to wait for a resource. 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.
<code>runAsUser</code>	<code>name</code> : The name of the user being impersonated in this job.
<code>scheduleName</code>	<code>name</code> : The schedule name that launched this job – this field differs from <code>liveSchedule</code> in that it is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
<code>start</code>	<code>date</code> : The time when this job began executing.

Object Type: `job`

Description: A *job* is a Commander structure associated with invoking a procedure. A new job is created each time a procedure begins to execute. 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. Commander retains job information after the job completes so you can examine what occurred.

Property Name	Description
<code>status</code>	<p>Possible values are:</p> <p><code>pending</code> - the job was created, but it is waiting for prerequisite steps to complete</p> <p><code>runnable</code> - the job step is waiting for a resource</p> <p><code>scheduled</code> - the job was assigned a resource, but the command has not started running</p> <p><code>running</code> - the job/job step is running a command on the assigned resource</p> <p><code>completed</code> - the job/job step has completed</p>
<code>totalWaitTime</code>	The total sum of license, resource, a precondition, and workspace wait times job steps were restricted and had to wait to process.
<code>workspaceWaitTime</code>	The sum of time all job steps had to wait for a workspace.

[\[back to top\]](#)

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>abortStatus</code>	<code>string</code> : If set, indicates the step was aborted. Values will be either <code>ABORT</code> or <code>FORCE_ABORT</code> - indicating how the step was aborted.
<code>abortedBy</code>	<code>name</code> : The user who issued the "abort."

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>actualParameters</code>	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>alwaysRun</code>	<code>boolean</code> : If true, this step runs even if the job is aborted before the step completes. Note that a "force abort" will abort an <code>alwaysRun</code> step.
<code>assignedResourceName</code>	<code>name</code> : The name of the resource assigned to this step by the step scheduler.
<code>broadcast</code>	<code>boolean</code> : If true, this step runs on all resources in a pool.
<code>combinedStatus</code>	Provides more inclusive step status output - the resulting query output may contain up to three sub-elements: <code>status message properties</code>
<code>command</code>	<code>string</code> : This property specifies the command this step runs.
<code>condition</code>	<code>string</code> : If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
<code>conditionExpanded</code>	<code>boolean</code> : The result of the expansion on the step condition.
<code>createTime</code>	<code>date</code> : The time when this object was created.

Object Type: <code>jobStep</code> Description: A <i>jobStep</i> is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a job .	
Property Name	Description
<code>delayUntil</code>	For a step that was rescheduled due to a resource or workspace problem, this is the next time the step will be eligible to run.
<code>elapsedTime</code>	number : The number of milliseconds between the start and end times for the <code>jobStep</code> .
<code>errorCode</code>	errorCode : This property appears when the outcome is <code>error</code> and displays the error code, identifying which error occurred.
<code>errorHandling</code>	<p>This is the error handling policy copied from the procedure step and indicates how the step responds to errors:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</p> <p><code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure.</p> <p><code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run.</p> <p><code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps.</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>ignore</code> - Continues as if the step succeeded.</p>
<code>errorMessage</code>	string : When the outcome is <code>error</code> , this property displays an error message description.
<code>exclusive</code>	boolean : If true, this step acquires and retains its resource exclusively.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>exclusiveMode</code>	<code>string</code> : Possible values are: none, job, step, call
<code>exitCode</code>	<code>number</code> : This is this step's exit code.
<code>external</code>	<code>boolean</code> : If "true", the job is external.
<code>finish</code>	<code>date</code> : The time when this job step completed.
<code>hostName</code>	<code>name</code> : The name of the host where this step was invoked (copied from the resource)
<code>job</code>	<code>id</code> : This is the ID number of the job that owns the job step. The string representation of the job is its name, so <code>\$(job)</code> evaluates to the job name, but <code>\$(job/foo)</code> is also legal and refers to the foo property of the job.
<code>jobId</code>	<code>id</code> : This is this job's ID number, which is a UUID.
<code>jobName</code>	<code>name</code> : This is the name of this step's job.
<code>jobStepId</code>	<code>id</code> : This is this step's ID number.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>licenseWaitTime</code>	The length of time this job step had to wait to process because the license limit was reached or exceeded.
<code>liveProcedure</code>	<code>string</code> : 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.

Object Type: jobStep

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
liveProcedureStep	name : This property shows the current procedure name for the procedure step from which this job step was created – if the procedure step was renamed since the job was launched, this will be the procedure step's new name, and if the procedure step was deleted, this will be null.
logFileName	name : The name of the log file produced by this step, relative to the job's workspace directory.
modifyTime	date : The time when this object was last modified.
outcome	string : The overall result of the job - possible values are: <i>success</i> , <i>warning</i> , <i>error</i> , or <i>skipped</i>
owner	name : The person (user name) who created the object.
parallel	boolean : If true, this step runs in parallel with other adjacent steps also marked to run in parallel.
postExitCode	number : This is this step's post processor exit code.
postLogFileName	name : The log file name produced by this step's post processor.
postProcessor	string : The post processor name used to gather information about this step.
precondition	string : By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated. A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>procedureName</code>	<code>name</code> : The name of the procedure that contains this <code>jobStep</code> .
<code>projectName</code>	<code>name</code> : The name of the project that contains this <code>jobStep</code> .
<code>propertySheet</code>	<code>reference</code> : propertySheet
<code>releaseExclusive</code>	<code>boolean</code> : A Boolean value indicating whether this step should release its resource upon completion.
<code>releaseMode</code>	<code>string</code> : Possible values are: none, release, releasetojob
<code>resourceName</code>	<code>name</code> : The name of the resource or pool this step should use to run on.
<code>resourceSource</code>	This property indicates whether the resource for the job step is explicitly specified by the step or was defaulted from the procedure: procedure or <code>procedureStep</code> .
<code>resourceWaitTime</code>	The length of time this job step stalled because it could not get a resource. 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.
<code>retries</code>	<code>number</code> : This is a number - the number of retries before the request times out.
<code>runAsUser</code>	<code>name</code> : The name of the user being impersonated in this job step.
<code>runnable</code>	<code>date</code> : The time when the step became runnable.
<code>runTime</code>	<code>number</code> : The number of milliseconds the step command spent running on a resource.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>shell</code>	name : The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a " {0} " marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
<code>start</code>	date : The time when this job step began executing.
<code>status</code>	<p>Possible values are:</p> <p><code>pending</code> - the job step was created, but it is waiting for prerequisite steps to complete</p> <p><code>runnable</code> - the job step is waiting for a resource</p> <p><code>scheduled</code> - the job step was assigned a resource, but the command has not started running</p> <p><code>running</code> - the job/job step is running a command on the assigned resource</p> <p><code>completed</code> - the job/job step has completed</p>
<code>stepName</code>	name : This is this step's name.
<code>subprocedure</code>	name : The nested procedure name to call when this step executes.
<code>subproject</code>	name : This property appears if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)
<code>timeLimit</code>	number : The maximum length of time this step is allowed to run.
<code>timeout</code>	date : The time when this job step will be automatically aborted if it has not yet completed.

Object Type: `jobStep`

Description: A *jobStep* is an instance of a step that ran - a single invocation of a command on a resource or a single call to a subprocedure within the context of a [job](#).

Property Name	Description
<code>totalWaitTime</code>	The sum of resource, workspace, and license wait times for this job step.
<code>waitTime</code>	number : The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).
<code>workingDirectory</code>	name : The name of this step's working directory.
<code>workspaceName</code>	name : The workspace name used by the <code>jobStep</code> object.
<code>workspaceWaitTime</code>	The time this job step had to wait because no workspace was found or available.

[\[back to top\]](#)

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the Commander system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The Commander server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
category	(currently not used)
container	<code>string</code> : Typically, this is the type and name of the workflow or job with a corresponding ID.
containerName	<code>name</code> : The name of the container.
containerType	<code>string</code> : The type of object the log entry pertains to (for example, procedure, job step, step).
deleted	<code>byte</code> : The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (which means "deleted" is not set).
logEntryId	<code>id</code> : The log entry Commander-generated ID.
message	<code>string</code> : The message text will either be informational or display the warning or error message. The message may contain important information about a resource or workspace issue.
principal	<code>string</code> : The user or project principal from the session that was active when the event occurred.

Object Type: logEntry

Description: A *logEntry* is an object containing various types of information available in a log file generated from anywhere in the Commander system.

The *Event Log* can be configured for auto-deletion. By default, Event Log retention is 30 days. The Commander server automatically marks old log entries for deletion and the background deleter cleans out old log entries.

To change the default settings, go to Administration > Server > Settings and modify the Event log retain time and the Maximum background delete delay fields

Note: Setting the "retain" period to "0" disables the automatic deletion mechanism, allowing you to use an external cleanup script to implement a custom cleanup policy.

Property Name	Description
severity	<code>string</code> : Severity can be either TRACE, DEBUG, INFO, WARN, ERROR
subject	<code>string</code> : The object associated with the message.
subjectName	<code>name</code> : The name of the object associated with the message.
subjectType	<code>string</code> : (similar to <code>container</code>) Refers to the object the event concerns. This may be the same as the container, or it may be a different object that is related to the event in some manner.
time	<code>string</code> : The time the event was logged.

[\[back to top\]](#)

Object Type: procedure

Description: A *procedure* describes a series of actions to be performed on one or more resources. A procedure contains steps and properties and can define formal parameters.

Property Name	Description
acl	reference: acl
createTime	date : The time when this object was created.
credentialName	name : The name of the credential assigned to this job step.
description	string : A user-specified text description of the object.
jobNameTemplate	name : The template used to name jobs created from this procedure.
lastModifiedBy	name : This shows who (generally, a user name) last modified this object.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
procedureId	id : This is this procedure's ID number.
procedureName	name : This is this procedure's name.
projectName	name : The name of the project that contains this procedure.
propertySheet	reference: propertySheet
resourceName	name : The name of the resource or pool this procedure should use to run on.
workspaceName	name : The workspace name used by the procedure object.

[\[back to top\]](#)

Object Type: project

Description: A *project* is an object used in Commander to organize information. A project contains procedures, schedules, credentials, and properties.

Property Name	Description
acl	<code>reference:</code> <code>acl</code>
createTime	<code>date:</code> The time when this object was created.
credentialName	<code>name:</code> The name of the credential assigned to this project.
deleted	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
description	<code>string:</code> A user-specified text description of the object.
lastModifiedBy	<code>name:</code> This shows who (generally, a user name) last modified this object.
modifyTime	<code>date:</code> The time when this object was last modified.
owner	<code>name:</code> The person (user name) who created the object.
pluginName	<code>name:</code> The name of the plugin associated with this project.
projectId	<code>id:</code> This is this project's ID.
projectName	<code>name:</code> The name of the project.
propertySheet	<code>reference:</code> <code>propertySheet</code>
resourceName	<code>name:</code> The name of the resource.
workspaceName	<code>name:</code> This is the workspace name used by the project.

[\[back to top\]](#)

Object Type: property

Description: A *property* is a string value with a name. Properties can have arbitrary names and values. You can attach properties to any object in the Commander system, such as a project, procedure, or job. After a property is created, it is stored in the Commander database. You can retrieve and/or modify the value later, and you can delete properties you no longer need. Properties provide a flexible and powerful mechanism to manage data about your builds.

Note: The name "properties" is NOT a valid property name.

Property Name	Description
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
expandable	<code>boolean</code> : If set to true, the property value will undergo string expansion when it is retrieved.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
canonicalPath	<code>string</code> : The path that specifies the property object.
propertyId	<code>id</code> : This property's ID number.
propertyName	<code>name</code> : This property's name.
propertySheetId	<code>reference</code> : <code>propertySheet</code> – If the property is a nested property sheet, the <code>propertySheet</code> refers to the nested sheet.
value	<code>string</code> : The property or actual parameter's value - if this property is a string property.

[\[back to top\]](#)

Object Type: `propertySheet`

Description: A property value can be a simple string or a nested *propertySheet* containing its own properties. Property sheets can be nested to any depth, which allows you to create hierarchical collections of information.

Most objects have an associated property sheet that contains "custom properties" created by user scripts.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheetId</code>	<code>id</code> : This property sheet's ID.

[\[back to top\]](#)

Object Type: `repository`

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on 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.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description for this object.

Object Type: repository

Description: A *repository* is an object that stores artifact versions. This object primarily contains information on 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.

Property Name	Description
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : <code>propertySheet</code>
repositoryDisabled	<code>boolean</code> : Determines whether the repository is disabled. Default is "false".
repositoryId	<code>id</code> : The Commander-generated ID number of the repository.
repositoryIndex	<code>number</code> : The order of the repository, within a list of repositories.
repositoryName	<code>name</code> : The name of the repository.
url	<code>string</code> : The server URL is in the form <code>protocol://host:port/</code> . Typically, the repository server is configured to listen on port 8200 for <code>https</code> requests, so a typical URL looks like <code>https://host:8200/</code> .
zoneName	<code>name</code> : The name of the zone where this repository is or will reside.

[\[back to top\]](#)

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
acl	<code>reference</code> : acl
agentState	<code>string</code> : Specific information about an agent, including the state of the agent. Possible values are: unknown alive down
artifactCacheDirectory	<code>string</code> : The directory on the agent host where retrieved artifacts are stored.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
exclusiveJobId	<code>id</code> : The ID number of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobName	<code>name</code> : The name of the job that owns this resource, which occurs when one of the job's steps requests exclusive use of the resource for the duration of the job.
exclusiveJobStepId	<code>id</code> : The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.
exclusiveJobStepName	<code>name</code> : 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.
gateways	The gateway name(s) associated with this resource.

Object Type: RESOURCE

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
hostName	name : The computer name or IP address for the machine containing the Commander agent for this resource.
hostOS	string : The full name of the host operating system, plus its version. However, if this host is a proxy, "proxied" appears as the host description/name.
hostPlatform	string : Examples for "platform" are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, "proxied" appears as the hostPlatform name.
lastModifiedBy	name : This shows who (generally, a user name) last modified this object.
lastRunTime	date : The most recent time a job step ran on the resource.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
pools	string : 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 : The port number for this resource.
propertySheet	reference : propertySheet
proxyCustomization	string : This property displays the customization of the Commander proxy agent.

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
proxyHostName	<code>name</code> : The name of the Commander agent being used to proxy to another agent, the proxy target.
proxyPort	<code>number</code> : The port number of the Commander agent being used to proxy to another agent, the proxy target.
proxyProtocol	<code>name</code> : The name of the proxy agent protocol used for communication from the Commander proxy agent to the proxy target - default is SSH.
repositoryNames	A "new line" separated list of repository names.
resourceDisabled	<code>boolean</code> : A Boolean value indicating whether this resource was disabled.
resourceId	<code>id</code> : This is this resource's ID.
resourceName	<code>name</code> : The name of the resource or pool.
shell	<code>name</code> : The shell used to execute the step's commands on a resource. If no shell was defined on a step, the shell defined on the resource is used, or if no shell was defined on the resource, a default shell is used. For shells: The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
stepCount	<code>number</code> : The current number of executing steps on this resource.
stepLimit	<code>number</code> : This property specifies the maximum number of steps that can run on this resource at one time.

Object Type: resource

Description: A *resource* is associated with a server machine available to ElectricCommander for running steps. A resource has a logical name in addition to a host name. A resource can be associated with one or more pools. The resource may be a "proxy target" machine that communicates with the Commander server through a Commander agent proxy machine.

Property Name	Description
trusted	<code>boolean</code> : If "true", this agent is trusted.
useSSL	<code>boolean</code> : A Boolean value indicating whether this resource uses SSL for communication.
workspaceName	<code>name</code> : The workspace name used by this resource.
zoneName	<code>string</code> : The name of the zone where this resource resides.

[\[back to top\]](#)**Object Type: resourcePool**

Description: A *resource pool* is a container for a group of resources.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
autoDelete	<code>boolean</code> : If "true", the resource pool is deleted when the last resource is removed or deleted.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
lastResourceUsed	<code>name</code> : The name of the most recently used resource from the pool.

Object Type: resourcePool

Description: A *resource pool* is a container for a group of resources.

Property Name	Description
modifyTime	<code>date</code> : The time when this object was last modified.
orderingFilter	<code>string</code> : 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.
owner	<code>name</code> : The person (user name) who created the object.
propertySheet	<code>reference</code> : <code>propertySheet</code>
resourcePoolDisabled	<code>boolean</code> : A Boolean value indicating whether this resource pool was disabled.
resourcePoolId	<code>id</code> : This resource pool's ID number.
resourcePoolName	<code>name</code> : The name of the resource pool.

[\[back to top\]](#)

Object Type: resourceUsage

Description: Provides usage information for all resources in the system.

For any job step running on a resource, there is a resource usage record that contains the ID and name of the job, job step, and resource.

Property Name	Description
jobId	<code>id</code> : This is this job's ID number, which is a UUID.
jobName	<code>name</code> : The name of the job.
jobStepId	<code>id</code> : The ID number of the job step.
jobStepName	<code>name</code> : The name of the job step.

Object Type: resourceUsage

Description: Provides usage information for all resources in the system.

For any job step running on a resource, there is a resource usage record that contains the ID and name of the job, job step, and resource.

Property Name	Description
licenseWaitTime	The time this job step had to wait because no license was found or available.
resourceId	<code>id</code> : The Commander-generated resource ID number.
resourceName	<code>name</code> : The name (or names) of the resource.
resourcePoolId	<code>id</code> : The Commander-generated resource pool's ID number.
resourcePoolName	<code>name</code> : The name of the resource pool.
resourceUsageId	<code>id</code> : The unique ID of the resource usage record.
resourceWaitTime	The time this job step had to wait because no resource was found or available. 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.
waitReason	Possible values are: <code>license</code> , <code>resource</code> , or <code>workspace</code> .
workspaceWaitTime	The time this job step had to wait because no workspace was found or available.

[\[back to top\]](#)

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
actualParameters	<p>reference: propertySheet An <i>actualParameter</i> 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.</p> <p>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.</p>
beginDate	<p>string: The first day on which the schedule is active, in the form YYYY-MM-dd. For example, "2009-06-25"</p>
createTime	<p>date: The time when this object was created.</p>
credentialName	<p>name: The name of the credential being used for impersonation when the schedule launches a job.</p>
description	<p>string: A user-specified text description of the object.</p>
endDate	<p>string: The end of the range of dates the schedule is active, in the form: YYYY-MM-dd. The actual end date is not included in the range. For example, "2009-06-25"</p>
interval	<p>string: A floating point number that represents the time to wait between invocations of the schedule.</p>
intervalUnits	<p>string: These are the units to use when interpreting the <i>interval</i> value. The <i>units</i> can be <code>hours</code>, <code>minutes</code>, <code>seconds</code>, or <code>continuous</code>.</p>
lastModifiedBy	<p>name: This shows who (generally, a user name) last modified this object.</p>
lastRunTime	<p>date: The last time a job was launched by a schedule.</p>

Object Type: <code>schedule</code> Description: <i>Schedules</i> are used to execute procedures and determine when specific procedures run.	
Property Name	Description
<code>misfirePolicy</code>	string: The policy the schedule uses when it is unable to launch a job at the scheduled time: <code>ignore</code> - wait until the next scheduled time to run <code>runOnce</code> - run immediately, then resume normal scheduling
<code>modifyTime</code>	date: The time when this object was last modified.
<code>monthDays</code>	string: This property specifies which days of the month this schedule should run - shown as a space-separated list of numbers (1-31).
<code>owner</code>	name: The person (user name) who created the object.
<code>priority</code>	string: Values can be <code>low</code> , <code>normal</code> (default), <code>high</code> , or <code>highest</code>
<code>procedureName</code>	name: The name of the procedure that contains this <code>schedule</code> .
<code>projectName</code>	name: The name of the project that contains this <code>schedule</code> .
<code>propertySheet</code>	reference: propertySheet
<code>scheduleDisabled</code>	boolean: A Boolean value indicating whether this schedule was disabled.
<code>scheduleId</code>	id: This schedule's ID number.
<code>scheduleName</code>	name: This property displays this schedule's name and differs from the <code>liveSchedule</code> property found under a job in that the <code>liveSchedule</code> property is written at job creation time only, and not changed, even if the schedule is renamed or deleted.
<code>startTime</code>	string: The time of day when this schedule should start running, in the form: <code>HH:mm:ss</code>

Object Type: schedule

Description: *Schedules* are used to execute procedures and determine when specific procedures run.

Property Name	Description
stopTime	string: The time of day when this schedule should stop running, in the form: HH:mm:ss
timeZone	string: A Java-compatible time zone string.
weekDays	string: This property specifies which days of the week this schedule should run. This is a space-separated list of MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY SUNDAY. (Note: These are not localized strings.)

[\[back to top\]](#)

Object Type: server

Description: This is the Commander server. There is only one such object in your database.

Read-only server properties

Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricCommander server location.

`/server/hostname` - returns the Commander server name

`/server/hostIP` - returns the Commander server IP address

These properties are especially useful for building a URL link to an ElectricCommander web page. For example, a link in an email notifier that links back to a Job Details page.

A sample link: `https://$[/server/hostname]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
acl	reference: <code>acl</code>
createTime	date: The time when this object was created.
hostIP	string: The server's IP address.

Object Type: `server`

Description: This is the Commander server. There is only one such object in your database.

Read-only server properties

Two read-only server properties are available for use in a step, a script, or an email notifier, which can be used to identify the ElectricCommander server location.

`/server/hostname` - returns the Commander server name

`/server/hostIP` - returns the Commander server IP address

These properties are especially useful for building a URL link to an ElectricCommander web page. For example, a link in an email notifier that links back to a Job Details page.

A sample link: `https://$[/server/hostname]/commander/jobDetails.php?jobId=$[jobId]`

Property Name	Description
<code>hostname</code>	<code>name</code> : Generally refers to the computer name or IP address for the machine containing the Commander server.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>serverId</code>	<code>id</code> : The server's ID number.
<code>serverName</code>	<code>string</code> : The Commander server's name - defaults to "server".

[\[back to top\]](#)

Object Type: state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (from the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
active	<code>boolean</code> : If "true", the state of the workflow is active.
actualParameters	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
errorMessage	<code>string</code> : When the outcome is <code>error</code> , this property displays an error message description.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the <code>project</code> that contains this state.
propertySheet	<code>reference</code> : <code>propertySheet</code>
stateId	<code>id</code> : The Commander-generated ID number for the state.

Object Type: state

Description: A *state* object belongs to a workflow and corresponds to the state definition, including a pointer to the workflow, formal parameters (cloned from definition), expanded actual parameters (from the last time a transition was taken to this state), and the "process" which includes the procedure or workflow (cloned from definition), unexpanded actual parameters (cloned from definition, expanded and passed to the process on entry), and the "last run" entity reference.

Property Name	Description
stateName	name : The name of the state.
subjob	name : The name of the subjob.
subparameters	reference : propertySheet
subprocedure	name : The name of the nested procedure called when a step runs.
subproject	string : If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	name : Name of the starting state for the workflow launched when the state is entered.
subworkflow	name : The name of the subworkflow - a workflow called by another workflow.
subworkflowDefinition	name : The name of the subworkflow definition.
workflow	name : The name of the workflow.

[\[back to top\]](#)

Object Type: stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
acl	<code>reference: acl</code>
actualParameters	<code>reference: propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
createTime	<code>date</code> : The time when this object was created.
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the <code>project</code> that contains this state definition.
propertySheet	<code>reference: propertySheet</code>
startable	<code>boolean</code> : "True" means this state definition can be the initial state of an instantiated workflow.
stateDefinitionId	<code>id</code> : The Commander-generated ID number for the state definition.
stateDefinitionName	<code>name</code> : The name of the state definition.

Object Type: stateDefinition

Description: A *stateDefinition* is a named object that belongs to a workflow definition, which includes specifications for whether or not the state is "startable", formal parameters, notifications, and the process. A process includes a procedure or workflow, the starting state (for workflows only), and actual parameters.

Property Name	Description
subprocedure	<code>name</code> : The name of the nested procedure called when a step runs.
subproject	<code>string</code> : If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.
substartingState	<code>name</code> : Name of the starting state for the workflow launched when the state is entered.
subworkflowDefinition	<code>name</code> : The name of the subworkflow definition.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.

[\[back to top\]](#)

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>actualParameters</code>	<code>reference</code> : <code>propertySheet</code> An <code>actualParameter</code> is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
<code>alwaysRun</code>	<code>boolean</code> : If true, this step runs even if the job is aborted before the step completes. Note that a force abort will abort an <code>alwaysRun</code> step.
<code>broadcast</code>	<code>boolean</code> : If true, this step will run on all resources in a pool.
<code>command</code>	<code>string</code> : This property specifies the command this step runs.
<code>condition</code>	<code>string</code> : If this element is not present, the event is ALWAYS triggered. If specified, the event is triggered only if the value of the condition argument is TRUE; if not true, a boolean value of "false" or "0" was used. Condition arguments can be a literal, a fixed value string, or a string subject to property expansion.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>credentialName</code>	<code>name</code> : The credential name assigned to this step.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
errorHandling	<p>This is the error handling policy copied from the procedure step and indicates how the step responds to errors:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <p><code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</p> <p><code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure.</p> <p><code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run.</p> <p><code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps.</p> <p><code>ignore</code> - Continues as if the step succeeded.</p>
exclusive	<code>boolean</code> : If true, this step acquires and retains its resource exclusively.
exclusiveMode	<code>string</code> : Possible values are: none, job, step, call
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
logFileName	<code>name</code> : The name of the log file produced by this step, relative to the job's workspace directory.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
parallel	<code>boolean</code> : If true, this step runs in parallel with other adjacent steps also marked to run in parallel.
postLogFileName	<code>name</code> : This property displays the log file name produced by this step's post processor.
postProcessor	<code>string</code> : This property displays the post processor name that is used to gather information about this step. Typically, this is either <code>postp</code> or <code>postp <options></code> , or an appropriate command or path including options to a postprocessor available on the Commander server.

Object Type: step

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
precondition	<p>string: By default, if a step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \"0\" or \"false\" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p>
procedureName	name: The name of the procedure that contains this <i>step</i> .
projectName	name: The name of the project that contains this step.
propertySheet	reference: propertySheet
releaseExclusive	boolean: A Boolean value indicating whether this step should release its resource upon completion.
releaseMode	string: Possible values are: none, release, releasetojob
resourceName	name: The resource's name this step should use to run on.

Object Type: `step`

Description: A *step* includes a command or script executed on a single resource and is the smallest unit of work ElectricCommander understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *pool* of equivalent machines, in which case Commander 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, Commander 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.

A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure.

Property Name	Description
<code>shell</code>	<code>name</code> : The shell used to execute the step's commands on a resource. The script name is inserted into the command at the position of a "{0}" marker in the command, or appended as a final argument if no marker is present. A filename suffix adjacent to the marker will be appended to the script name. For more information on shells, see the Shell definition in the Step Help topic.
<code>stepId</code>	<code>id</code> : The step's ID number.
<code>stepName</code>	<code>name</code> : The step's name.
<code>subprocedure</code>	<code>name</code> : The nested procedure name to call when this step executes.
<code>subproject</code>	<code>name</code> : This property is displayed if the subprocedure element is present and specifies the project to which the subprocedure belongs (by default, the current project is used.)
<code>timeLimit</code>	<code>number</code> : A floating point number that specifies the maximum length of time this step is allowed to run.
<code>timeLimitUnits</code>	The units to use when interpreting the time limit. Units can be <code>hours</code> , <code>minutes</code> , or <code>seconds</code> .
<code>workingDirectory</code>	<code>name</code> : The name of the step's working directory.
<code>workspaceName</code>	<code>name</code> : The workspace name used by this step.

[\[back to top\]](#)

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>
<code>actualParameters</code>	<code>reference</code> : <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
<code>condition</code>	<code>string</code> : If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.
<code>createTime</code>	<code>date</code> : The time when this object was created.
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>index</code>	The numeric index of a transition that indicates the transition order in the state definition's transition list.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>projectName</code>	<code>name</code> : The name of the <code>project</code> that contains this transition.
<code>propertySheet</code>	<code>reference</code> : <code>propertySheet</code>
<code>stateName</code>	<code>name</code> : The name of the state.
<code>targetState</code>	<code>string</code> : The target state for the transition definition.
<code>transitionId</code>	<code>id</code> : The Commander-generated ID for the transition.

Object Type: transition

Description: A *transition* object belongs to a state and corresponds to the transition definition, and includes the target state (unexpanded actual parameters cloned from definition, expanded and passed to the target state on entry), the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`).

Property Name	Description
transitionName	<code>name</code> : The name of the transition.
trigger	<code>string</code> : Possible values are: <code>onEnter</code> - before any actions <code>onStart</code> - after a subjob or workflow is created <code>onCompletion</code> - after a subjob or workflow completes <code>manual</code> - when a user manually requests a transition
workflowName	<code>name</code> : The name of the workflow.

[\[back to top\]](#)**Object Type: transitionDefinition**

Description: A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (`onEnter|onStart|onCompletion|manual`)).

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
actualParameters	<code>reference</code> : <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter passed to the target state.
condition	<code>string</code> : If empty or non-zero, the transition is allowed to trigger. If set to "0", the transition is ignored.
createTime	<code>date</code> : The time when this object was created.

Object Type: transitionDefinition

Description: A *transitionDefinition* is a named object that belongs to a state definition, which includes the target state definition (with actual parameters to the target state, the Javascript condition, and the trigger (`onEnter`|`onStart`|`onCompletion`|`manual`)).

Property Name	Description
<code>description</code>	<code>string</code> : A user-specified text description of the object.
<code>index</code>	The numeric index of a transition that indicates the transition order in the state definition's transition list.
<code>lastModifiedBy</code>	<code>name</code> : This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date</code> : The time when this object was last modified.
<code>owner</code>	<code>name</code> : The person (user name) who created the object.
<code>projectName</code>	<code>name</code> : The name of the project that contains this transition definition.
<code>propertySheet</code>	<code>reference</code> : propertySheet
<code>stateDefinitionName</code>	<code>name</code> : The name of the state definition.
<code>targetState</code>	<code>string</code> : The target state for the transition definition.
<code>transitionDefinitionId</code>	<code>id</code> : The Commander-generated ID for the transition definition.
<code>transitionDefinitionName</code>	<code>name</code> : The name of the transition definition.
<code>trigger</code>	<code>string</code> : Possible values are: <code>onEnter</code> – before any actions <code>onStart</code> – after a subjob or workflow is created <code>onCompletion</code> – after a subjob or workflow completes <code>manual</code> – when a user manually requests a transition
<code>workflowDefinitionName</code>	<code>name</code> : The name of the workflow definition.

[\[back to top\]](#)

Object Type: USER

Description: *User* refers to any user currently logged into the Commander system or any user who may use Commander, but may not be currently logged in.

Property Name	Description
<code>acl</code>	<code>reference:</code> <code>acl</code>
<code>createTime</code>	<code>date:</code> The time when this object was created.
<code>email</code>	<code>name:</code> Displays this user's email address.
<code>fullUserName</code>	<code>name:</code> The user's real name.
<code>lastModifiedBy</code>	<code>name:</code> This shows who (generally, a user name) last modified this object.
<code>modifyTime</code>	<code>date:</code> The time when this object was last modified.
<code>mutable</code>	<code>boolean:</code> If true, the user is editable within ElectricCommander via the web UI or the <code>modifyUser</code> API.
<code>owner</code>	<code>name:</code> The person (user name) who created the object.
<code>propertySheet</code>	<code>reference:</code> <code>propertySheet</code>
<code>providerName</code>	<code>name:</code> The name of the <code>directory provider</code> that controls this user.
<code>userId</code>	<code>id:</code> A unique ID number for a user object.
<code>userName</code>	<code>name:</code> Displays this user's name and also appears in a group object and displays the name of a user who belongs to this group.

[\[back to top\]](#)

Object Type: workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting state, parameters to the initial state, "complete" - a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
acl	<code>reference</code> : <code>acl</code>
activeState	<code>string</code> : The name of the active state on the workflow object.
actualParameters	<code>reference</code> : <code>propertySheet</code> An <i>actualParameter</i> is an object that provides the value for a parameter that is passed to the target state when the transition is taken.
callingState	<code>string</code> : The full property path to the state that created this workflow.
callingStateId	<code>id</code> : The ID number for the full property path to the state that created this workflow.
completed	<code>boolean</code> : If "true", the workflow is completed and no additional transactions will be evaluated.
createTime	<code>date</code> : The time when this object was created.
deleted	<code>boolean</code> : The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
elapsedTime	The time this object ran from start to finish.
finish	<code>date</code> : The date/time when this object finished.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
launchedByUser	<code>string</code> : The name of the user or project principal that explicitly launched the workflow.
liveWorkflowDefinition	<code>name</code> : The current workflow definition name for the workflow definition from which the workflow was created.

Object Type: workflow

Description: A *workflow* object includes a pointer to the workflow definition, sets of states, the active state, the starting state, parameters to the initial state, "complete" - a boolean value determining whether or not the workflow is complete, and a log containing the transactions taken and user events.

Property Name	Description
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the project that contains this workflow.
propertySheet	<code>reference</code> : propertySheet
start	<code>date</code> : The date/time when this workflow began to run.
startingState	<code>string</code> : The initial state of the workflow.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.
workflowId	<code>id</code> : The Commander-generated ID for the workflow.
workflowName	<code>name</code> : The name of the workflow.

[\[back to top\]](#)**Object Type: workflowDefinition**

Description: A *workflowDefinition* object contains state and transition definitions, including the workflow name template (analogous to a job name template on a procedure), and ordered sets of state and transition definitions.

Property Name	Description
acl	<code>reference</code> : acl
createTime	<code>date</code> : The time when this object was created.

Object Type: workflowDefinition

Description: A *workflowDefinition* object contains state and transition definitions, including the workflow name template (analogous to a job name template on a procedure), and ordered sets of state and transition definitions.

Property Name	Description
description	<code>string</code> : A user-specified text description of the object.
lastModifiedBy	<code>name</code> : This shows who (generally, a user name) last modified this object.
modifyTime	<code>date</code> : The time when this object was last modified.
owner	<code>name</code> : The person (user name) who created the object.
projectName	<code>name</code> : The name of the <code>project</code> that contains this workflow definition.
propertySheet	<code>reference</code> : <code>propertySheet</code>
workflowDefinitionId	<code>id</code> : The Commander-generated ID for the workflow definition.
workflowDefinitionName	<code>name</code> : The name of the workflow definition.
workflowNameTemplate	<code>string</code> : Template used to determine the default names for workflows launched from a workflow definition.

[\[back to top\]](#)

Object Type: workspace

Description: ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and Commander automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
acl	<code>reference:</code> acl
agentDrivePath	<code>name:</code> The drive-letter based mount point for this workspace on Windows agents.
agentUncPath	<code>name:</code> The UNC path for this workspace on Windows agents.
agentUnixPath	<code>name:</code> The UNIX path for this workspace on UNIX agents.
createTime	<code>date:</code> The time when this object was created.
credentialName	<code>name:</code> The name of the credential being used when a resource connects to a network share while setting up the workspace.
description	<code>string:</code> A user-specified text description of the object.
lastModifiedBy	<code>name:</code> This shows who (generally, a user name) last modified this object.
local	<code>boolean:</code> If "true", this workspace is local.
modifyTime	<code>date:</code> The time when this object was last modified.
owner	<code>name:</code> The person (user name) who created the object.
propertySheet	<code>reference:</code> propertySheet

Object Type: workspace

Description: ElectricCommander provides each job step with an area on the disk it can use for "working files" and results. This disk area is called a job *workspace*. (The job workspace defaults to a directory under the workspace directory.) A job step can create whatever files it needs within its workspace, and Commander automatically places files such as step logs in the workspace.

Normally, a single workspace is shared by all steps in a job, but different steps within a job could use different workspaces. The location of the job step workspace is displayed on the Job Details page for the job under "Details" for the step.

Property Name	Description
<code>workspaceDisabled</code>	<code>boolean</code> : A Boolean value that indicates whether this workspace was disabled.
<code>workspaceId</code>	<code>id</code> : The workspace's ID number.
<code>workspaceName</code>	<code>name</code> : The workspace's name.
<code>zoneName</code>	<code>name</code> : The name of the zone where this workspace resides.

[\[back to top\]](#)

Object Type: Zone

Description: Commander provides the ability to create zones—often used to enhance security.

- 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—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the [Gateways](#) Help topic.

Property Name	Description
<code>acl</code>	<code>reference</code> : <code>acl</code>

Object Type: Zone

Description: Commander provides the ability to create zones—often used to enhance security.

- 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—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the [Gateways](#) Help topic.

Property Name	Description
createTime	date : The time when this object was created.
description	string : A user-specified text description for the object.
lastModifiedBy	name : This person (generally, a user name) last modified this object.
modifyTime	date : The time when this object was last modified.
owner	name : The person (user name) who created the object.
propertySheet	reference : propertySheet
resources	string : A space-separated list of resources contained in this zone.
zoneId	id : The Commander-generated ID for this zone.
zoneName	string : The name of the zone.

Property error codes

The following list provides error codes that may appear as a value within an `errorCode` property.

ABORTED

ACCESS_DENIED

AGENT_BAD_WORKINGDIR

AGENT_FAILED_CONNECT_BROKER

AGENT_FAILED_CREATE_FILE
AGENT_FAILED_CREATE_WORKSPACE
AGENT_FAILED_IMPERSONATION
AGENT_FAILED_MAP_DRIVE
AGENT_FAILED_PROXYTARGET_PING
AGENT_INCOMPATIBLE_VERSION
AGENT_INTERNAL_ERROR
AGENT_INVALID_CWD
AGENT_INVALID_MESSAGE
AGENT_INVALID_PING_TOKEN
AGENT_INVALID_WORKSPACE
AGENT_IO_CONNECTION_REFUSED
AGENT_IO_CONNECTION_RESET
AGENT_IO_ERROR
AGENT_IO_NO_ROUTE_TO_HOST
AGENT_IO_PORT_UNREACHABLE
AGENT_MALFORMED_XML
AGENT_NONABSOLUTE_PATH
AGENT_NONEXISTENT_DIR
AGENT_NONEXISTENT_FILE
AGENT_PING_FAILED
AGENT_STREAM_STOPPED
AGENT_TIMEOUT
AGENT_UNKNOWN_CMDID
AGENT_UNKNOWN_COMMAND
AGENT_WRONG_FILE_TYPE
BAD_AGENT_RESPONSE
BAD_LOGFILE_PATH
CANCELED
CIRCULAR_PROCEDURE_REFERENCE
CORRUPT_CREDENTIAL

DUPLICATE_JOB_NAME
EMPTY_SUBPROCEDURE
FAILED_JOB_RENAME
FORMAL_PARAMETER_ERROR
INTERNAL_ERROR
INVALID_EMAIL_HEADER
MY_EVENT_EXPANSION_ERROR
NO_EMAIL_HEADERS_SEPARATOR
NONEXISTENT_EMAIL_CONFIG
NONEXISTENT_PROCEDURE
NONEXISTENT_RESOURCE
NONEXISTENT_WORKSPACE
NOTIFIER_EXPANSION_ERROR
POST_PROCESSOR_ERROR
POST_PROCESSOR_OUTPUT_FAILURE
PROPERTY_REFERENCE_ERROR
RESOURCE_WITHOUT_HOSTNAME
SERVER_SHUTDOWN
TIMEOUT
UNKNOWN_HOST

Property - create new or edit existing property

Use this pop-up window to create or modify a custom property attached to an object.

Fill-in the fields as follows:

Field Name	Description
Name	<p>The property name can be an arbitrary text string, but unless you are an experienced Commander user avoid using slashes and brackets as a part of the property name.</p> <p>Note: The name "properties" is NOT a valid property name.</p>
Description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
Value	This value can be an arbitrary text string.
Expandable	"uncheck" this box if you do not want to allow property expansion.

Click **OK** to save your new or modified property information.

For more information on creating and using properties, see the [Properties](#) Help topic.

Nested Property Sheet

Use this pop-up window to define or modify a nested property sheet.

Fill-in the fields as follows:

Name - The property name can be an arbitrary text 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>` `` `
` `<div>` `<dl>` `` `<i>` `` `` `<p>` `<pre>` `` `<style>` `<table>` `<tc>` `<td>` `<th>` `<tr>` ``

Click **OK** to save your new or modified nested property sheet information.

For more information on creating and using properties, see the [Properties](#) Help topic.

Reports

If you have upgraded ElectricCommander, this page displays historical reports (Cross Project Summary, Daily Summary, and Resource Summary) from your previous Commander version. These reports are no longer being updated.

Click the Historic/web server local reports link to see any previously created custom reports.

- Cross Project Summary - this is a 30-day report summary
- Daily Summary - this is a daily report showing all the procedures that ran
- Resource Summary - this report shows resource usage for a specific day
- Historic/web server local reports

For more information, see the [Reports](#) Help topic.

Create a new report

Use this page to create a new report. Select one of the report-type tabs to begin.

Multiple Series Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	<p>This is your report title. Type over the default report name, choosing any unique name for your report.</p> <div>IMPORTANT: Do not use special characters in the report title. Some examples of special characters are : / ? # [] @ ! , \$ & ' () * + , ; =</div>
Saved Filter	<p>Project - Click your mouse inside this field to see a list of projects from which to make a selection.</p> <p>Filter - After selecting a project, this drop-down menu will contain any available filters for this project.</p>
Time Period	Use the drop-down menu to select the time period.
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.
Table column choices	<p>Chart Type - Use the drop-down menu to choose the chart type.</p> <p>Function - Use the drop-down menu to choose the function you need.</p> <p>Property Name - Use the default property value or delete the text and click your mouse in the blank field to see a list of possible properties.</p> <p>Display Name - You can choose a different unique Display Name.</p> <p>Stacked - Select this checkbox to see your report results "stacked" versus overlaid.</p> <p>The "X" icon - Click this icon to delete any row you no longer need.</p>

Field/Menu Name	Description
Add Series button	Select the Add Series button if you would like to create additional table entries for additional report information.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.
Table Options	Use the down-arrow to adjust the Time Grouping and see the default for the Time Column label.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays a box with a default schedule name that you can change if you choose. Click OK to go to the Job Details page.
Cancel button	Use this button to cancel the create report operation.

Category Report

Field and pull-down menu descriptions:

Field/Menu Names	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. IMPORTANT: Do not use special characters in the report title. Some examples of special characters are : / ? # [] @ ! , \$ & ' () * + , ; =
Saved Filter	Project - Click your mouse inside this field to see a list of projects from which to make a selection. Filter - After selecting a project, this drop-down menu will contain any available filters for this project.
Time Period	Use the drop-down menu to select the time period.

Field/Menu Names	Description
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.
Object Property	You can delete the default value and click your mouse inside the blank field to see a list of properties from which to choose.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click OK to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Count Over Time Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	This is your report title. Type over the default report name, choosing any unique name for your report. IMPORTANT: Do not use special characters in the report title. Some examples of special characters are : / ? # [] @ ! , \$ & ' () * + , ; =
Saved Filter	Project - Click your mouse inside this field to see a list of projects from which to make a selection. Filter - After selecting a project, this drop-down menu will contain any available filters for this project.
Time Period	Use the drop-down menu to select the time period.

Field/Menu Name	Description
Create thumbnail?	Check this box if you would like to make a thumbnail view of this report available on your Home page.
Object Type	Use the drop-down menu to select an object.
Chart Options	Use the down-arrow to adjust the Time Grouping and see the defaults for the X and Y axis.
Table Options	Use the down-arrow to adjust the Time Grouping and see the default for the Time Column label.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click OK to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Procedure Usage Report

Field and pull-down menu descriptions:

Field/Menu Name	Description
Report Title	<p>This is your report title. Type over the default report name, choosing any unique name for your report.</p> <div style="border: 1px solid black; background-color: #e6f2ff; padding: 5px;"> <p>IMPORTANT: Do not use special characters in the report title. Some examples of special characters are : / ? # [] @ ! , \$ & ' () * + , ; =</p> </div>

Field/Menu Name	Description
Project	<p>Click your mouse inside this blank field to see a list of projects. This does not have to be the current project.</p> <p>You can generate reports that show when one project's procedures call procedures in another project. You can also use the '%' wildcard, for example,</p> <p>Project: %</p> <p>or</p> <p>Project: D%</p>
Procedure pattern	Supply a SQL string match pattern for procedures to include in the report. The default % will use all procedures.
Advanced	Use the down-arrow to select a Locale or add a Credential. Delete the Resource if you choose, then click your mouse in the blank field to see a list of other resources you might prefer to use.
Run Report button	Use this button to go to the Job Details page to run the report.
Create Schedule button	This button displays box with a default schedule name that you can change if you choose. Click OK to go to the Job Details page.
Cancel button	Use this bottom to cancel the create report operation.

Resources

This page displays all resources known to this ElectricCommander server and provides easy access to resource configuration and all other resource management functionality.

- Each resource has a logical name—a unique name to distinguish this resource from other resource names.
- Each resource refers to an agent machine by its host name.
- Each resource can be assigned to one or more pools.
A *pool* is a group of interchangeable resources. For example, you might have a pool of Windows servers. If you name a pool in a procedure step, Commander is free to assign that step to any resource in the pool, which gives Commander the freedom to choose a lightly loaded resource. You can change the resources in a pool without having to modify any procedures that use the pool.
- Each resource can be assigned to a zone. A default zone is created during Commander installation and all resources, by default, are members of that zone until assigned to a different zone.
A *zone* is a collection of agents. Every agent, and all resources defined on that agent, will belong to one zone only. When Commander was installed, a *default* zone was created. If a zone is deleted, all agents in that zone are moved to the *default* zone. The Commander server resides in the default zone.
Note: The *default* zone **cannot** be deleted.
- Several resources can correspond to the same physical host or agent machine.
- When you specify a resource name in a procedure step, the step executes on that resource.

Commander supports two types of resources

- **Standard** - This resource type specifies a machine running the Commander agent on one of the supported agent platforms, as specified in the *ElectricCommander Installation Guide*.
- **Proxy** - This resource type requires SSH keys for authentication. You can create proxy resources (agents and targets) for Commander to use on numerous other remote platforms/hosts that exist in your environment.
 - Proxy *agent* - This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform. A proxy agent is a Commander agent, channeling to a proxy target. Proxy agents have limitations, such as the inability to work with plugins or communicate with ectool commands.
 - Proxy *target* - This is a machine on an unsupported platform that can run commands via an SSH server.

Note: If you have upgraded from a previous Commander release, you have already noticed the main table on this page has changed—more columns and more functionality were added. And with the addition of quick-access slide-out panels, you no longer have to navigate between multiple resource web pages—for example, to create or edit a resource.

Help topic "Best Practice": We recommend reviewing this Help topic in its entirety to become familiar with new resource management features and expanded functionality. Later, when using the Resources page, use the following quick links to go to the help section you need to review.

[Icons/links above the table](#)

[Table view - column descriptions](#)

[Grid view](#)

[Filters pane](#)

[New Resource panel](#)

[New Proxy Resource panel](#)

[Edit Resource panel](#)

[Edit Proxy Resource panel](#)

[Resource Details panel](#)

[Switching a non-trusted agent to trusted](#)

[Install or Upgrade Remote Agents](#)

Resource page information and functions

Notice the information immediately after the Resources title above the table - you can see how many licensed resources are currently in use.







Icons/links above the table

Hover your mouse over an icon to see available actions to select. All actions can be used for a single resource or numerous resources can be selected (use first column checkbox) to perform a bulk operation for any number of resources at the same time.

- Select one or more resources from the table, then select the icon to perform the task or bulk operation you need.



- Exception: Using the Action icon and selecting Create Resource or Create Proxy Resource opens a panel to create one of these new objects. **Note:** No bulk operation is available.

Icon / link name	Description
	<p>A multiple action icon:</p> <p>Create Resource - the New Resource panel opens. For information on filling in the fields, see the New Resource section below.</p> <p>Create Proxy Resource - the New Proxy Resource panel opens. For information on filling in the fields, see the New Proxy Resource section below.</p> <p>Install/Upgrade Resource(s) - opens a popup dialog series to supply information about the agents to install or upgrade. For more information, see the Install/Upgrade Remote Agents section below.</p> <p>Notes:</p> <ul style="list-style-type: none"> • If you are upgrading one or more agents/resources, the label on this option will be Upgrade Resource(s). • This option is available for Linux agents only. You cannot install or upgrade Windows agents using this process. • Important: If the EC-AgentManagement plugin is NOT installed, you will not see this option.
	<p>Copy Resource - makes a duplicate copy of one or more resources you select to copy. The new resource copies will appear in the table</p>
	<p>Delete Resource - deletes one or more resources, depending on the number of resources you select, using the first column checkboxes.</p> <p>Note: If this resource is on a "gateway agent", the gateway will be deleted.</p>
	<p>Ping Resource - to see if one or more resources are available.</p> <p>When ElectricCommander "pings" a resource, it sends a message to the agent [for the resource] to make sure the agent is alive and running a version of software compatible with the Commander server. After the ping completes (or fails), the page refreshes to reflect the current resource state.</p> <p>Note: A gateway must exist before you can ping a resource in a remote zone.</p>
	<p>Enable Resource - enables one or more resources, depending on the number of resources you selected, using the first column checkboxes.</p>
	<p>Disable Resource - disables one or more resources, depending on the number of resources you selected, using the first column checkboxes.</p>








Icon / link name	Description
	<p>A multiple function icon:</p> <ul style="list-style-type: none"> Clicking this icon with no resources selected displays the message: "Select some resources to edit" Clicking this icon with resources selected provides several options: <ul style="list-style-type: none"> Add to Pool - adds the selected resources to a pool. Remove from Pool - shown only if selected resources have at least one pool in common. If selected resources belong to one common pool only, this link becomes, "Remove from Pool <name>". Move to Zone - moves the selected resources to a zone. Set Trusted - this option is not displayed if selected resources are "trusted" already. If not, all selected resources will be modified to <i>trusted</i>. Unset Trusted - shown only if all selected resources are trusted. If not, selected resources will be modified to <i>untrusted</i> status.
  or	Select this icon to add this Resources page to your Home page for one-click access to return to this page. If the icon is "yellow", the page is accessible from your Home page already.
New Search link	Use this link to go to the Define Search page if you need to search for a particular object or set of objects. For example, job, resource, artifact, workflow, and so on. Use the "back" button to return to the Resources page.






Table view - column descriptions

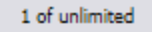
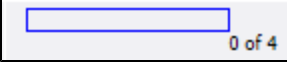
Not all resource configuration information is displayed within the table, but you can see all other information about the resource when you edit the resource or access other slide-out panels. For example, the action to see or add access control privileges to a resource is available on the Edit Resource panel or from the **Access Control** link on that panel.

Note: Most column headings are "sortable". Click on a column heading to re-sort the information in that column.

Column name / icon	Description
	<p>Select this box to enable the corresponding resource in that row for a single or bulk operation. For example, deleting or enabling multiple resources at the same time is a bulk operation.</p> <p>Note: if you select this icon in the table header row, it will select or deselect all resources in the table.</p>

Column name / icon	Description
	<p>Indicates agent status. If an agent machine is not available, any resources associated with that host will not be available.</p> <p>Note: If an agent is down, the Commander server is not able to provide additional information for the cause.</p>
Name	<p>The name of the resource. The icon to the left of the resource name indicates whether this resource is <i>enabled</i> or <i>disabled</i>. These icons are: Enabled  or Disabled </p> <p>If a resource is disabled, no job steps are assigned to it. If a step requests a particular resource and that resource is disabled, the step waits until the resource becomes enabled again. If the resource is in a pool, Commander uses any resource in the pool to satisfy the request for the resource. Other actions:</p> <ul style="list-style-type: none"> To change the enabled/disabled icon, select the checkbox (first column) in the row for one or more resources, then select the enabled or disabled icon above the table. Select multiple resources if this is a bulk change. <p><i>Enable/Disable operation change: If you are an existing Commander user who upgraded to Commander v4.2, the enable/disable column with "checkboxes" to change a resource status has been replaced with the process described above.</i></p> <p>Note: You can change the resource enable/disable specification from the Edit Resource panel too, but this is an individual operation for one particular resource.</p> <ul style="list-style-type: none"> Click on a resource name to see the Resource Details panel - this panel displays how your resources is configured currently. To make changes, click the Edit link to go to the Edit Resource panel.
Pools	<p>This is a space-separated list of pool names where this resource is a member.</p> <p>Note: Spaces are not allowed within a pool name.</p>

Column name / icon	Description
Job Status	<p>Indicates the current job status with the following icons:</p> <p> <i>Running job</i> - a job is still running, which means the resource is currently being used for one or more jobs. The names of all jobs using the resource are displayed next to the icon.</p> <p> <i>Success</i> - the job running on this resource completed successfully.</p> <p> <i>Error</i> - the job running on this resource encountered an error and may or may not have completed. One or more error messages will be listed in this column.</p> <p> <i>Warning</i> - the job running on this resource encountered a warning and may or may not have completed. One or more warning messages will be listed in this column.</p> <p> <i>Busy</i> - Commander is attempting to refresh the job status, but the resource is not responding or the UI cannot parse the response. Changing the value of 'Agent Host Name' to the IP address can sometimes resolve this issue.</p>
Description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>.</p>
Zone	The name of the zone where this resource is a member.
HTTPS & Host	The "Connection Type" you specified when this resource was created.

Column name / icon	Description
Step Load	<p>Resource <i>load</i> is indicated by one of the following:</p> <p></p> <p>- this notation indicates 1 step is using this resource and an unlimited number of steps can use this resource at the same time. If 5 steps are using this resource, you would see "5 of unlimited".</p> <p></p> <p>- this notation indicates that no steps are running on this resource and the step limit is set to 4. The bar indicates graphically how close you are to the step limit.</p> <p>Note: Use the Create/Edit Resource panels to define the number of steps that can run on a resource. Setting the Step Limit to "0" or leaving the field blank defaults to NO step limit.</p>

Grid view



Two icons above the Filters pane:

- The first icon (on the left) is the default *Table view* icon.
- The second icon (on the right) displays the resource *Grid view*.

Use the drop-down menus to filter the Grid view:

- Grouping - Select the resource "grouping" you want to see—choose Pool, Job, Operating System, Platform, Proxy Agent, Version, or Zone.
For example, if you have multiple resource pools defined and chose Pool, you will see groups labeled with your various pool names and a group (No Pool) for resources not included in a pool.

- Coloring - Select what you want to view in each group—choose Platform, Step Count/Limit, Status, Held Exclusive, Version, or Operating System to sort your group.

Note: Colors are dynamically assigned per the view you select. Within a single view, the same color "square" implies similar information.

- Hovering your mouse over a grid "square" provides more detailed information about that resource.

Filters pane

Use the Filters pane to filter resources shown in either the Table or Grid view. The fields in the Filter pane correspond to columns in the Table view.

Supply Filter criteria as follows:

Actions / Field	Description
Save Filters	Click this link to save your filter criteria. A dialog box is displayed so you can name your filter. For easy, repeat access, filters are saved in a "saved filters" box above the link. When you select a saved filter name, the Filter pane fields populate automatically.
Reset	Use this link to clear all entries in the Filter pane.
"Quick Search" field	Supply any text for your search. For example, enter a host name. You do not need to enter a full name or text string - this field filters on a partial text entry, sometimes 2-3 characters are all you might need.
Status	The status for the agent machine.
"drop-down" menu	Default is both enabled and disabled resources. Use the drop-down arrow to choose enabled or disabled only.
"Pools" field	List one or more pool names, separated by a space. (No spaces are allowed within a pool name). Pool names must be entered exactly - a shorten version of the pool name will not yield a successful result.
"Hosts" field	List one or more full host names as shown in the Table view "HTTPS & Host" column (each name separated by a space). Supplying a partial name will not yield a successful result.
Step Limit "drop-down"	Choose the step limit corresponding to your desired search.
Proxy Agent "drop-down" menu	Select No if not searching for a proxy agent or Yes if you are trying to locate a proxy agent.
Filter button	Select this button after you complete the fields/selections to define your filter criteria.

New Resource panel

To define a new standard resource, fill-in the fields as follows:

Field	Description
Name	<p>Supply a unique name for this resource. Do not use "spaces" in a resource name.</p> <p>This is the name used to select the resource for a job step—it need not be the same as the host name for the machine.</p>
Description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
Agent Host Name	<p>Supply the domain name or IP address of the agent machine for this resource. This is the name all other machines in this agent/resource's zone will use to communicate with this agent host.</p>
Agent Port Number	<p>Supply the port number to use when connecting to the agent for the resource, Default port is 7800.</p>
Default Workspace	<p>Supply the workspace name or leave blank to use the local, default workspace. The workspace specification can be overridden at the project, procedure, or step level.</p> <p>If you specify a workspace name here, it will be used as the default for all job steps that run on this resource. See the Workspaces Help topic for more information.</p>
Pool(s)	<p>A space-separated list of arbitrary names indicating the pools associated with this resource. Do not use "spaces" in a pool name.</p>

Field	Description
Default Shell	<p>The name of the shell program used to execute the step's commands on a resource. For example, using <code>sh</code> or <code>cmd /c</code>, the agent saves the command block to a temporary file with a <code>.cmd</code> extension and runs it: <code>sh foo.cmd</code> or <code>cmd /c foo.cmd</code>.</p> <p>If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:</p> <p>Windows: <code>cmd /q /c "{0}.cmd"</code> UNIX: <code>sh -e "{0}.cmd"</code></p> <p>When you specify a shell (in the step or resource), and omit the <code>cmd-file</code> marker, the agent notices the omission and takes the correct action. For example: a user specifies <code>sh -x</code>. The agent converts this to <code>sh -x "{0}.cmd"</code>.</p> <p>Two alternate forms of shell syntax where Commander uses a "marker," <code>{0}</code>, as a placeholder for the command file argument:</p> <ul style="list-style-type: none"> • <code><myShell> {0} <potential extra shell args></code> In this example, the command file is not meant to be the last argument in the final command line. For example, <code>mysql -e "source {0}"</code> This shell example runs the <code>mysql</code> command against this step's command containing <code>sql</code>. • <code><myShell> {0} <.file extension> <potential extra shell args></code> In this example, the shell requires the command file to end in an extension other than <code>.cmd</code>. For example, <code>powershell "& '{0}.ps1'"</code> This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands. <p>Notes:</p> <ul style="list-style-type: none"> • When the agent parses the shell, it will parse the extension as everything after <code>{0}.</code> until it sees a space or non-alphanumeric character. • If your script uses International characters (non-ascii), add the following block to the top of your <code>ec-perl</code> command block: <pre>use utf8; ElectricCommander::initEncodings</pre>

Field	Description
Step Limit	<p>The maximum number of steps that can execute simultaneously on this resource—a step limit applies to a particular resource only, not to its underlying host.</p> <p>For example, if you define two resources, resource1 with a step limit of 5 and resource2 with a step limit of 1, both specifying the same host machine, it is possible for a total of 6 steps to execute simultaneously on the underlying host.</p> <p>The Resource Details panel displays this information.</p> <p>Note: Setting the Step Limit to "0" or leaving the field blank defaults to NO step limit.</p>
Artifact Cache Directory	<p>The directory on this resource's agent host from which artifact versions are retrieved and made available to job steps.</p> <p>Supply an absolute path to the resource containing this cache directory.</p>
Zone	<p>The name of the zone where this resource is or will be a member—a zone is a top-level network containing one or more mutually accessible resources.</p>
Repository Names	<p>A "new-line" delimited list of repository names for this resource, if needed. When a step attempts to retrieve artifact versions from a Commander repository, it queries repositories specified here. If no repositories are specified, it will "fallback" to the default repository.</p>
Connection Type	<p>Use the drop-down menu to select your Connection Type:</p> <ul style="list-style-type: none"> • HTTP - choose this option if you are not using SSL and the agent does not need to be "trusted". • HTTPS - choose this option if you are using SSL, but this agent does not need to be "trusted". • Trusted HTTPS - choose this option if you are using SSL and this agent must be "trusted". A trusted agent is "certificate verified"—the agent verifies the server or "upstream" agent's certificate. <p>Note: For information about the certificate used for trusted agents, see the Commander-Installed Tools, "eccert" section, Help topic.</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • <i>trusted</i> - the Commander server verifies the agent's identity using SSL certificate verification. • <i>untrusted</i> - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.

Field	Description
Enabled	<p>"Check" this box to enable this resource.</p> <p>When this box is checked, the resource is enabled, which means job steps can be assigned to it. If a job step requests a pool containing this resource, the step executes on a resource in that pool.</p> <p>If disabled, and this is the only resource that satisfies a particular job step, the step's execution is delayed until the resource is re-enabled. If a resource is disabled while job steps are running on it, the running steps continue to completion but no new steps are assigned to that resource.</p>

Click **OK** to save your settings and see your new resource listed in the table.

New Proxy Resource panel

Reminder - proxy agent vs. proxy target:

- Proxy *agent* - This is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported platform.
- Proxy *target* - This is a machine on an unsupported platform that can run commands via an SSH server.

To define a new proxy resource, fill-in the fields as follows:

Field	Description
General	
Name	<p>Supply a unique name for this resource. <i>Do not use</i> "spaces" in a resource name.</p> <p>This is the name used to select the resource for a job step—it need not be the same as the host name for the machine.</p>
Description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
Proxy Agent Host Name	<p>Supply the domain name or IP address of the proxy agent machine corresponding to this resource.</p>

Field	Description
Proxy Agent Port Number	Supply the port number to use when connecting to the agent for the resource.
Default Workspace	<p>Supply the workspace name for leave blank to use the local, default workspace.</p> <p>If you specify a workspace here, it will be used as the default for all job steps that run on this resource. See the Workspaces Help topic for more information.</p>
Pool(s)	Supply a space-separated list of arbitrary names, indicating the pools associated with this resource. Do not use "spaces" in a pool name.
Step Limit	<p>Specify the maximum number of steps that can execute simultaneously on this resource—a step limit applies to a particular resource only, not to its underlying host.</p> <p>For example, if you define two resources, <code>resource1</code> with a step limit of 5 and <code>resource2</code> with a step limit of 1, both specifying the same host machine, it is possible for a total of 6 steps to execute simultaneously on the underlying host.</p>
Zone	The name of the zone where this resource will be a member.
Connection Type	<p>Use the drop-down menu to select your Connection Type:</p> <ul style="list-style-type: none"> • HTTP - choose this option if you are not using SSL and the agent does not need to be "trusted". • HTTPS - choose this option if you are using SSL, but this agent does not need to be "trusted". • Trusted HTTPS - choose this option if you are using SSL and this agent must be "trusted". A trusted agent is "certificate verified"—the agent verifies the server or "upstream" agent's certificate. <p>Note: For information about the certificate used for trusted agents, see the Commander-Installed Tools, "eccert" section, Help topic.</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • <i>trusted</i> - the Commander server verifies the agent's identity using SSL certificate verification. • <i>untrusted</i> - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.

Field	Description
Enabled	<p>"Check" this box to enable this resource.</p> <p>When this box is checked, the resource is enabled, which means job steps will be assigned to it. If a job step requests a pool containing this resource, the step executes on a resource in that pool.</p> <p>If disabled, and this is the only resource that satisfies a particular job step, the step's execution is delayed until the resource is re-enabled. If a resource is disabled while job steps are running on it, the running steps continue to completion but no new steps are assigned to that resource.</p>
Proxy Target	
Proxy Target Host Name	Supply the domain name or IP address of the proxy target machine corresponding to this resource.
Proxy Target Port Number	Supply the port number to use when connecting to the proxy target host or leave blank to use the default. The default port is 22, the SSH server port.

Field	Description
Proxy Target Default Shell	<p>The name of the shell program used to execute commands on this [the proxy target] resource—may be overridden at the step level.</p> <p>This shell will be used to execute the step's commands on a resource. For example, using <code>sh</code> or <code>cmd /c</code>, the agent saves the command block to a temporary file with a <code>.cmd</code> extension and runs it: <code>sh foo.cmd</code> or <code>cmd /c foo.cmd</code>.</p> <p>If you do not specify a shell on a step, at step run-time the server looks at the resource shell. If a resource shell is not set, the shell line used by the agent is platform dependent:</p> <p>Windows: <code>cmd /q /c "{0}.cmd"</code> UNIX: <code>sh -e "{0}.cmd"</code></p> <p>When you specify a shell (in the step or resource), and omit the <code>cmd-file</code> marker, the agent notices the omission and takes the correct action. For example: a user specifies <code>sh -x</code>. The agent converts this to <code>sh -x "{0}.cmd"</code>.</p> <p>Two alternate forms of shell syntax where Commander uses a "marker," <code>{0}</code>, as a placeholder for the command file argument:</p> <ul style="list-style-type: none"> • <code><myShell> {0} <potential extra shell args></code> In this example, the command file is not meant to be the last argument in the final command line. For example, <code>mysql -e "source {0}"</code> This shell example runs the <code>mysql</code> command against this step's command containing <code>sql</code>. • <code><myShell> {0} <.file extension> <potential extra shell args></code> In this example, the shell requires the command file to end in an extension other than <code>.cmd</code>. For example, <code>powershell "& '{0}.ps1'"</code> This shell example runs Microsoft PowerShell against this step's command containing PowerShell commands. <p>Notes:</p> <ul style="list-style-type: none"> • When the agent parses the shell, it will parse the extension as everything after <code>{0} .</code> until it sees a space or non-alphanumeric character. • If your script uses International characters (non-ascii), add the following block to the top of your <code>ec-perl</code> command block: <pre>use utf8; ElectricCommander::initEncodings</pre>

Field	Description
Proxy Customizations	This is proxy-specific customization data. Default is "none". See the "Commander-Installed Tools" Help topic (ecproxy section) for more information.

Click **OK** to save your settings.

Edit Resource panel

All of the fields on this panel are the same as those on the New Resource panel:

- All information you previously provided to create this resource is filled-in for you.
- You can change any information already supplied.
- You can add information in any blank fields.

Click **OK** to save your modifications.

IMPORTANT: If you want to change the Connection Type, using the UI by itself is insufficient. You must also use the `ecconfigure` utility to change the protocol. Run: `ecconfigure --agentProtocol [http|https]`

Edit Proxy Resource panel

All of the fields on this panel are the same as those on the New Proxy Resource panel:

- All information you previously provided to create this resource is filled-in for you.
- You can change any information already supplied.
- You can add information in any blank fields.

Click **OK** to save your modifications.

Resource Details panel

This panel displays how this resource is configured with the Commander server. Some of the information supplied is linked to its source. For example, the zone name for this resource is a link to its Zone Details panel.

- The name of the resource is shown below the panel title.
- The resource description is shown under the resource name.
- Links at the top of the panel:
 - Use the **View Usage** link to go to the Job Step Search Results page.
 - Use the **Edit** link to modify the resource specifications.
 - Use the **Access Control** link [at the top of the panel] to specify or view access privileges for this resource.
- The tabs:

- General - by default, the panel opens to display the "general" resource configuration.
- Properties - select this tab to add properties to this resource or see existing resource properties if already configured.

Custom Resource Properties

You can define custom properties for any Commander object. For example, if configuration information varies from resource to resource, use custom properties to store this information and then reference it from procedure steps where it is needed.

For example, if you have a pool of test machines with test hardware in a different location on each machine, you could define a property named "testLocation" on each resource, then pass this property to procedure steps using a reference such as `$/myResource/testLocation`.

Switching a non-trusted agent to trusted

If you have upgraded to Commander v4.2 or later, none of your existing agents were converted to "trusted" during the upgrade process. By default, the *local* agent is not trusted during a new or upgrade installation.

After an upgrade, if you want to use a trusted configuration, any **pre-existing** agent/host machines must be manually switched to "trusted".

Manually switching a non-trusted agent to trusted status

Perform the following steps to change the agent status to trusted in your environment and in the customer's site.

1. Enter `ectool --server <COMMANDER_SERVER_IP> login admin` to log into the server and save the session ID.
2. Enter `eccert initCA` to initialize the server Certificate Authority (CA).
A new CA key and certificate are created.
3. Enter `eccert --server <COMMANDER_SERVER_IP> initAgent --remote --force` to install the agent with a self-signed certificate that needs to be overwritten.
4. Restart the agent.


If you do not restart the agent, it tries to use a previously cached certificate if one exists. The old certificate is invalid because a new one was just created.

5. In the Commander UI, ping the agent.

For information about the Electric Commander Certificate Authority (CA) and the certificates configured in Commander Server and Commander Agent installations, see [Commander Installed Tools](#).

Install or Upgrade Remote Agents



Select this icon, , to install or upgrade host machines you want to use as agent machines for your Commander resources. A popup dialog series opens for you to supply information about the hosts to install or

upgrade.

- If no hosts are selected in the Resources Table view, you will be installing host names you supply.
- If host names are listed already in the Resources Table view, selected one or more to perform an upgrade operation.

Notes:

- The dialog screens you see for an Install operation will be somewhat different from those for an Upgrade operation, depending on the options you choose.
- You can install or upgrade Windows, Linux, Solaris, Solaris x86, HP-UX, or MacOS hosts.
- When you install the agent software on a Solaris host, install it in a directory path that has less than 70 characters. .

Prerequisites:

- You must have an artifact repository server installed.
- The target machines must be running the SSH daemon.
- The user you select must be set up for passwordless *sudo* privileges.
For example, in the `/etc/sudoers` file, you must add "`<username> ALL=(ALL) NOPASSWD:ALL`"

IMPORTANT: If the EC-AgentManagement plugin is *not* installed, the remote install/upgrade option will not be available and you cannot install/upgrade agents remotely from the Commander user interface. During a new Commander installation, this plugin is installed automatically.

- For each remote agent machine you want to upgrade, make sure PasswordAuthentication=yes in `/etc/ssh/sshd_config`.
Example:

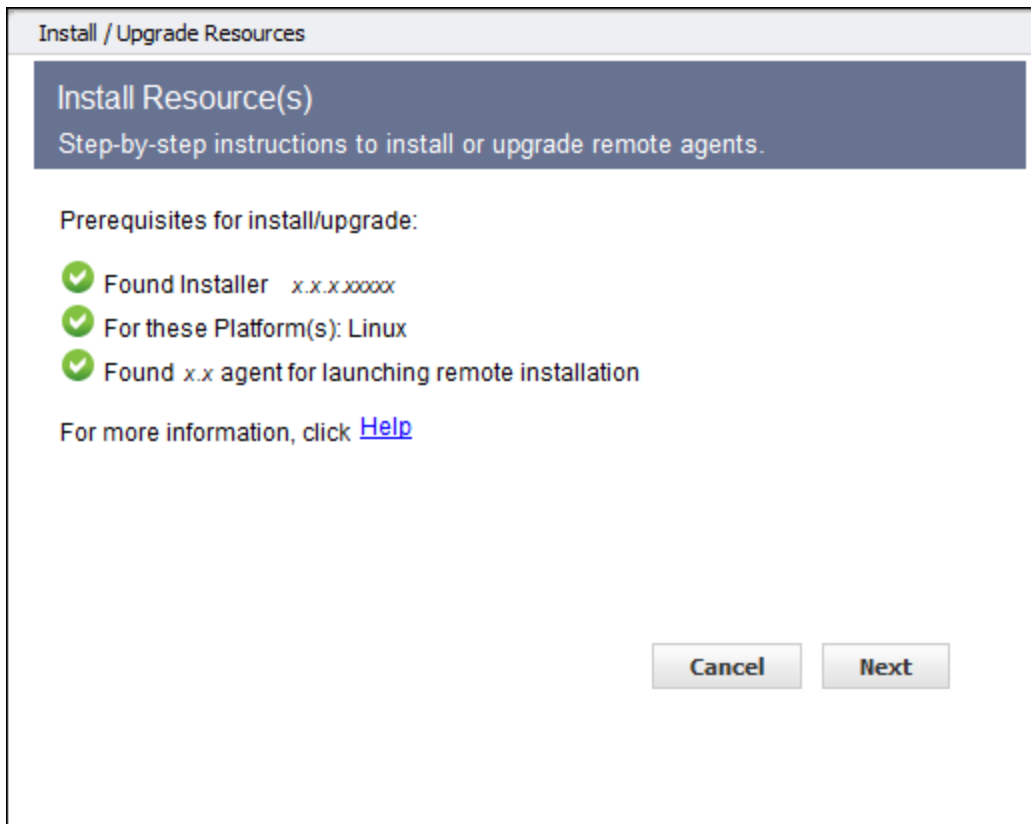
```
# To disable tunneled clear text passwords, change to no here!  
PasswordAuthentication yes
```


PasswordAuthentication=no is the default on SUSE platforms, so you must edit `/etc/ssh/sshd_config`. Restart the sshd service after changing the value.
- On Windows, WMI has the following additional prerequisites:
 - The target machine must belong to the same domain as the driving agent machine.
 - The agent service user of the driving agent must have administrator rights on the target machine.
- On Unix, SSH has the following additional prerequisite:
 - The user account on the target machine must have password-less sudo configured for running the installer with root privileges.

Using the Install or Upgrade Remote Agents dialog

The Commander server runs several verification checks before you begin an Install/Upgrade operation, and when you see the following screen with all green "checkmark" icons, you can proceed.

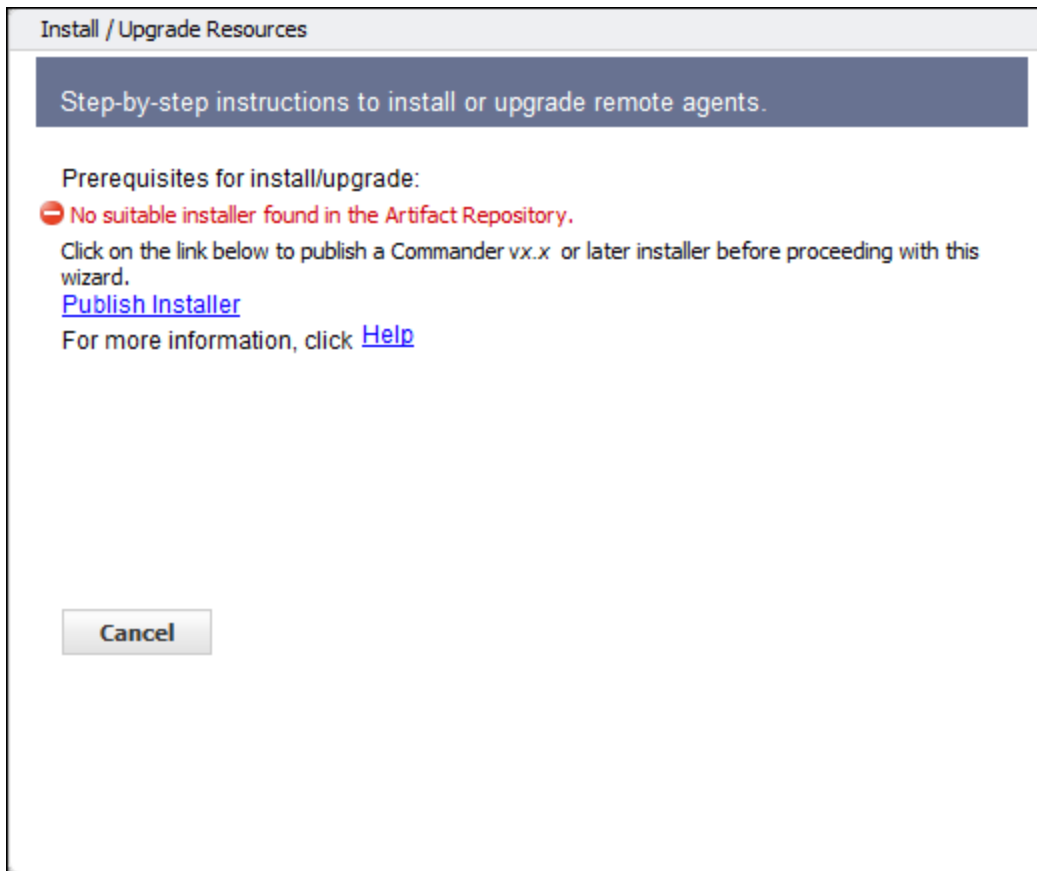
Note: Depending on your selections, you will see variations in some to the following screen examples.



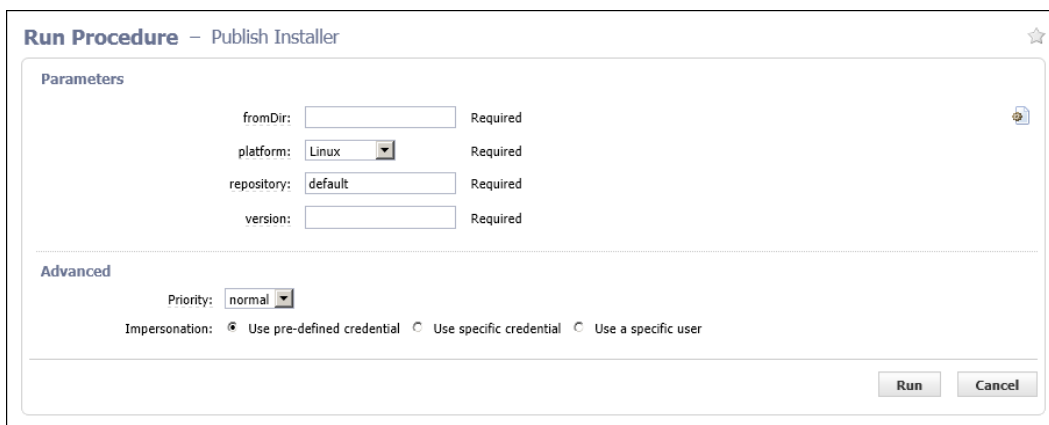
Click **Next** to continue.

However, if the plugin is installed but a Commander v4.2 or later installer is not found in the artifact repository, you will see an error screen similar to the example below.

Click the **Publish Installer** link.



After clicking the **Publish Installer** link, the next screen is displayed.



On this screen, fill-in the fields as follows:

Parameters

`fromDir` - Supply the path to the directory containing the installer on the Commander server machine. For example, `C:/<temp>` for Windows or `/var/tmp` for Linux.

`platform` - Use the drop-down menu to make a selection

`repository` - Use `default` to use the repository installed during the Commander installation, or type-over `default` and supply a different repository name

`version` - Supply the build version for your Commander install, for example, "5.0.0.56390". Your build version can be obtained from the name of the Commander installer you downloaded from the Electric Cloud FTP site.

Advanced

(Not required to complete the Publish Installer run procedure)

When the Publish Installer procedure runs, you can see the job status on the Job Details page.

For verification:

Go to the Artifacts page > select `com.electriccloud:installer`, which displays the Artifact Details page. This page displays all of your published installers.

Return to the Resources page and begin again...

Assuming you now have all green "check mark" icons on the first dialog screen, and you clicked **Next**, the following dialog screen is displayed.

Install / Upgrade Resources

Target Options
Supply list of hostnames to install

Zone:

Install to Host:

Name Templates:

Template:

Fill-in the fields as follows:

Zone - If you have not defined additional zones, the `default` zone is used and you will not see this field. If you have multiple zones defined already, use the drop-down menu to choose the zone where you are installing or upgrading agents. ***You can install or upgrade agents to only one zone at a time.*** A functioning gateway must exist before an agent can be installed into the non-default zone. So installing the first agent into a zone must be done manually.

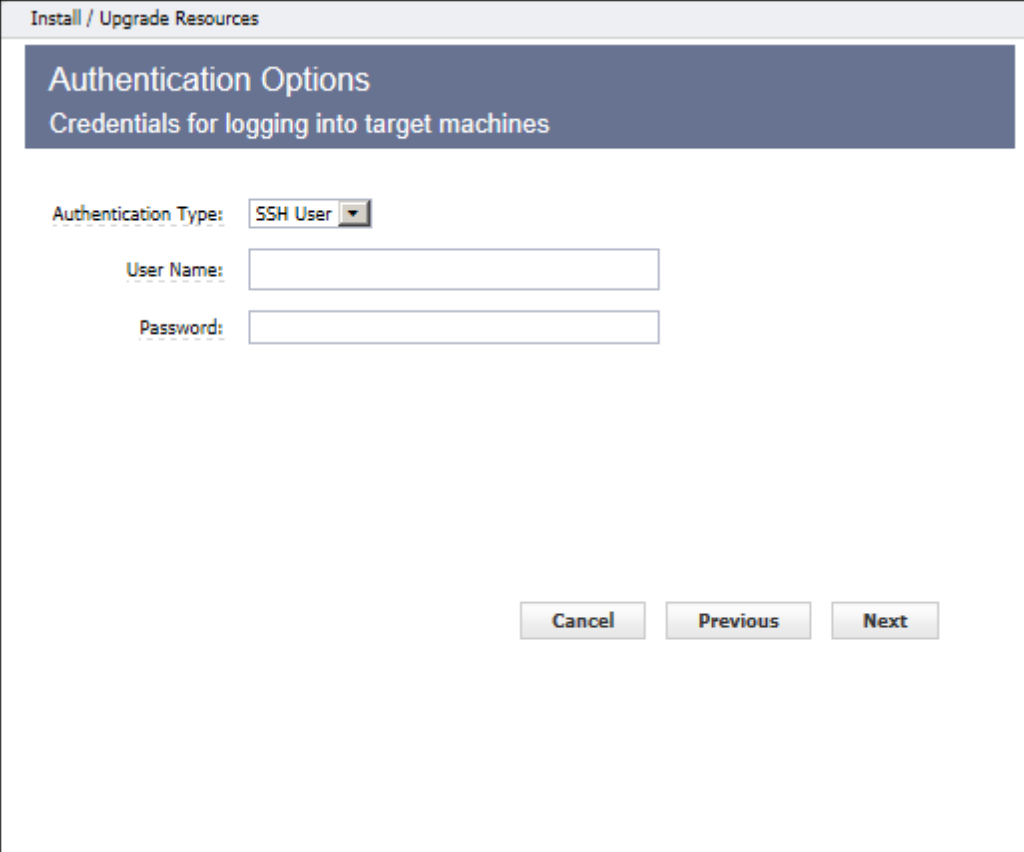
Install to Host - (required) Supply one or more host names or IP addresses—a space-separated list. Individual host names ***cannot*** contain spaces. If this is an upgrade operation, host names or IP addresses are pre-populated in this field from the resources you selected for upgrade on the Resources page - Table view, first column.

Name Templates - Use the drop-down menu to choose a simple template for setting the names of newly created resources.

Template - You can edit this field only if you selected Custom in the Name Template drop-down menu. This field undergoes property expansion in a global context and is scanned for special tokens `{name}` and `{counter}`.

Driving Resource - This resource performs all actions for installing or upgrading a target host on behalf of the server. For a Windows target, this resource must be a Windows host in order for WMI communication to work properly.

Click **Next** to continue to the next dialog screen.



Install / Upgrade Resources

Authentication Options
Credentials for logging into target machines

Authentication Type: SSH User ▼

User Name:

Password:

Cancel Previous Next

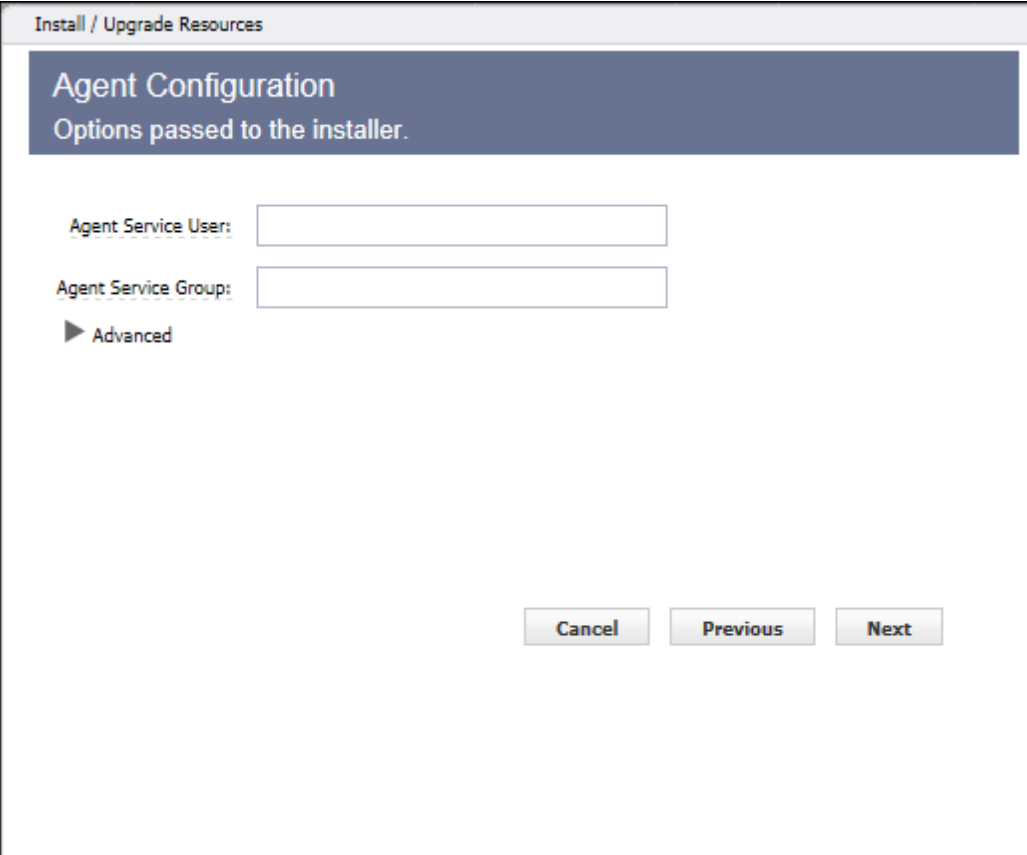
Fill-in the fields as follows:

Authentication Type - use the drop-down menu to choose SSH User, SSH Key, or WMI:

- If SSH User -
 - User Name - (required) Supply a user name.
 - Password - (required) Supply a password for the user name.
- If SSH Key -
 - User Name - (required) Supply a user name.
 - Public Key Path - (required) The path to the SSH public key file.
 - Private Key Path - (required) The path to the SSH private key file.
 - Passphrase - (optional) The passphrase for unlocking the private key file.

- If WMI -
 - No additional information is needed

Click **Next** to continue to the next dialog screen.



Install / Upgrade Resources

Agent Configuration
Options passed to the installer.

Agent Service User:

Agent Service Group:

☐ Advanced

Cancel Previous Next

Note: You will not see this screen if you are *upgrading* hosts.

For installing hosts, fill-in the fields as follows:

Agent Service User - (required) the user the Commander agent "runs as".

Agent Service Group - (required) the group the Commander agent "runs as".

Selecting "Advanced" is optional - Select this option to define non-default values.

- If you did not select Advanced, click **Next** to continue (and skip the next dialog screen example).
- If you selected Advanced, see the next dialog screen example.

Install / Upgrade Resources

Agent Configuration

Options passed to the installer.

Agent Service User:

Agent Service Group:

▼ Advanced

Agent Port:

Install Directory:

Data Directory:

Trusted Agent: ☐

Cancel Previous Next

Advanced options: (optional)

Agent Port - Specify the listener port for the agent.

Install Directory - Specify the path to the install directory.

Data Directory - Specify the path to the directory where your modified Commander files are stored (configuration and log files).

Trusted Agent - Select this check box and all agent machines being installed or upgraded will be "trusted".

For more information, see [Switching a non-trusted agent to trusted](#).

Click **Next** to continue to the summary dialog.

Your summary dialog will be similar to the following example, but it will contain the values you specified.

Note: If you are installing or upgrading more than 5 hosts, you will see a quantity number and NOT a space-separated list of host names

Install / Upgrade Resources

Ready to Install

Review the settings before starting the install.

Number of Host(s):

gerrit jira-test linplugin1

Zone:

alpha

Template:

Hostname/IP

UserName:

build

Password:

[PROTECTED]

Agent Service User:

qe

Agent Service Group:

build

Trusted Agent:

Untrusted

Select Finish to run the install.

Cancel

Previous

Finish

Click **Finish** to start the install or upgrade and the Job Details page is displayed.

When install/upgrade is complete, you can return to the Resources page to see your newly installed or upgraded hosts listed in the Table or Grid view.

To verify a resource version, click a resource name (in the Resources page Table view) to open the Resource Details panel for that resource.

Resource Pools

This page displays all resource pools currently available to ElectricCommander.

- To create a new resource pool, click the **Create Resource Pool** link.
- To find an existing resource pool, use the **New Search** link.

Column descriptions

Column Name	Description
Pool Name	This is the name you supplied for this resource pool. Click on a Pool Name to go to the Resource Pool Details page.
Enabled	If this box is "checked", this resource pool is enabled.
Auto Delete	If there is a "checkmark" in this column, the pool will be deleted when the last resource is deleted or removed.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Actions	<p>Use the Copy link to make an exact duplicate of an existing resource pool, then select the copy to go to the Resource Pool Details page to change the resource pool name or add or remove resources from the pool.</p> <p>Use the Delete link to delete the resource pool on the same row.</p> <p>Note: Deleting a resource pool does not delete the resources referenced by the pool.</p>

Resource Pool - create new or edit existing pool

To create a new resource pool

Fill-in the fields as follows:

Field Name	Description
Name	Supply a name for the resource pool. This name must be unique within a list of multiple resource pool names and cannot be the same as any resource name.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Ordering Filter	<p>Leave this field blank to use the default. When assigning a resource pool to a job step, the default resource selection algorithm is as follows:</p> <ul style="list-style-type: none"> Resources in the pool are sorted by the following criteria, and the first 'usable' resource is selected. <ul style="list-style-type: none"> Descending exclusive job ID Descending exclusive job step ID Ascending step count Random The implications of this sort order (when combined with the 'usable' criteria below) are: <ul style="list-style-type: none"> Prefer a resource that is already <code>call</code>-exclusive to an ancestor of the job step Prefer a resource that is already <code>job</code>-exclusive to the step's job Prefer a resource with a lower step count All things being equal, choose a resource at random <p>For more information on overriding the default algorithm, see the section below.</p>

Field Name	Description
Auto Delete	<p>If you select this checkbox, the pool is deleted when the last resource is deleted or removed.</p> <ul style="list-style-type: none"> • The <code>autoDelete</code> property flag is automatically set to "true" if the pool was created on the Create Resource page, as a side-effect of the resource API calls. • By default, resource pools created explicitly on the Create Resource Pool page, have the <code>autoDelete</code> flag set to false—the checkbox is "unchecked".
Enabled	Select this checkbox to enable the resource pool.

Click **OK** to save the information to create a new resource pool. You will see your new resource pool listed on the Resource Pools page.

Note: Resource pools cannot be nested, which means if the name of a resource pool is added to a resource pool, it will be ignored.

A resource pool can override the default algorithm by providing a JavaScript fragment in the Ordering Filter (`orderingFilter`) intrinsic property. The JavaScript environment for this fragment is:

- A variable named `resourceList` – an array of the resources in the pool, sorted as described above
- A variable named `resourcePool` – the pool object
- The job step being scheduled is configured as the context object for property resolution

The evaluation result is expected to be an array of resources, in the order they should be considered by the scheduler.

Before evaluation, the JavaScript fragment is wrapped as a function, to make it possible to use the `return` statement in the script.

It is possible to return any resources in the result, not just those grouped in a pool, so this feature enables true dynamic resource selection. Resources are still subject to the "usable" criteria. See the "Unusable resources" section at the end of this help topic.

Examples for the Ordering Filter field:

- Sort by resource name in descending order:

```
return resourceList.sort(function(a,b) {
return b.resourceName.localeCompare(a.resourceName);
});
```

- Ignore the default resources and return different resources:

```
var result = new Array();
result[0] = resources["resource1"];
```

```
result[1] = resources["resource2"];  
return result;
```

To edit an existing resource pool

Use this page to modify any specifications for the resource pool.

Click **OK** to save the information and update the resource pool.

Unusable resources

Resources, whether in a pool or not, are not considered usable if any of the following are true:

- The resource does not exist
- The resource is disabled
- The resource does not have a host name
- The agent where the resource is associated is not reachable
- The number of steps running on the resource is equal to the resource step limit
- The resource is exclusive to a different job using the job `exclusiveMode`
- The resource is exclusive to a step in a different job using the step or call `exclusiveMode`
- The resource is exclusive to a different step in the same job using the call `exclusiveMode` and the other step is not an ancestor of the step
- The resource is exclusive to a different step in the same job using the step `exclusiveMode`
- The job step does not have execute privileges on the resource
- The resource assignment exceeds the license limit (if any)

Resource Pool Details

This page displays information for the resource pool you selected, including a summary of the specifications supplied to create the resource pool, current resources assigned to this pool, and the ability to add or remove resources.







Links and actions at the top of the page

- **Edit** - use this link to go to the Edit Resource Pool page.
- **Access Control** - use this link to go the Access Control page for this resource pool.
- The "star" icon allows you to save this information to your Home page.
- The "curved arrow" icon returns you to the Resource Pools page.
- **Pagination** - Use the "previous" and "next" arrow icons to view the previous or next project. The numbers between the arrow icons display the number of projects you can view and the first number indicates which project [in the list] you are viewing.

The "tabbed" sections

The tabs at the top of the table allow you to select the type of information you want to see. The Resource Pool Details page opens with the Resources tab highlighted, so the first table you see is the Resources table. See the following screen example.

This table displays the resources currently included in this pool. The name of the pool ("custom" in our example) follows the Resource Pool Details page title.

Resource Pool Details – custom     6/8  

Description: Use this pool for QA testing only.

Ordering Filter:

Auto Delete: yes

Enabled: yes

Resources | Properties

Resource Name	Status	Enabled	Description	Host	Proxy Agent	Version	HTTPS	Step Limit	Add Resource to Pool	Actions
agent1	✓	✓		mustang3		3.5.3.35364	✓	1	Ping Remove	
agent10	✓	✓		mustang3		3.5.3.35364	✓	10	Ping Remove	
agent2	✓	✓		mustang3		3.5.3.35364	✓	2	Ping Remove	
agent3	✓	✓		mustang3		3.5.3.35364	✓	3	Ping Remove	

The summary section at the top of the table contains information previously defined when the pool was created:

Description - The text previously supplied for this object when it was created.

Ordering Filter - The Javascript ordering filter (see the [New Resource Pool](#) help topic for more information) or "empty" if no filter was applied.

Auto Delete - When "yes" is specified, the pool will be deleted when the last remaining resource is removed or deleted from the pool.

Enabled - "yes" specifies this pool is enabled for use.

Link(s) at the top of the table

Add Resources to Pool - Click this link to see a pop-up dialog to supply a resource name to add to this pool. You can type a name or select from a list of existing resource names. If other resources are already configured, you will see a list from which to select a resource. Click **OK** to add new resources to the pool.

Column descriptions

Column Name	Description / Actions
Resource Name	The name of the resource. Click on a resource name to go to the Edit Resource page.
Status	Indicates the current status of the resource.
Enabled	Indicates whether the resource is enabled. If a resource is disabled, no job steps will be assigned to it. Commander will use other resources within the pool to satisfy requests for the pool.
Description	The text previously supplied for this object when it was created.
Host	Steps assigned to run on this resource will use this host.
Proxy Agent	The name or IP address of the proxy agent machine.
Version	The version of the Commander agent installed on this resource.
HTTPS	A "checkmark" indicates HTTPS was selected when this resource was created.
Step Limit	The maximum number of steps that can execute simultaneously on this resource.
Actions	Ping - Use this link to check the status of the resource. Remove - Use this link to remove a resource from this pool.

Properties tab

This tab provides a table listing all properties for the resource pool and includes the following functionality:

Links at the top of the table

- To create a new property for this resource pool, click the **Create Property** or **Create Nested Sheet** link.
- To view or change privileges on the property sheet, click the **Access Control** link.

Column descriptions

Column Name	Description / Action
Property Name	The name of the property. Click on a property name to edit that property.
Value	Indicates the value assigned to this property
Description	A text description previously supplied for your reference only.
Actions	Delete - Use this link to delete this property.

The following screen is an example of the Properties table.

Resources	Properties			
				Create Property Create Nested Sheet Access Control
Property Name	Value	Description	Actions	
postSummary			Delete	

Zones

This page displays zones currently available to ElectricCommander and provides other zone operations—create, edit, delete, and so on.

- 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—agents in one zone cannot directly communicate with agents in another zone.
- A *default* zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).
- Each zone can have one or more "gateway agents", which you define. Gateway agents are used for communication from one zone to another zone. For more information, see the [Gateways](#) Help topic.
- Every agent, and all resources defined on that agent, can belong to one zone only.
- Within a zone, agents can be either *trusted* or *untrusted*.
 - *trusted* - the Commander server verifies the agent's identity using SSL certificate verification.
 - *untrusted* - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.

Links and actions at the top of the table

- To create a new zone, click the **Create Zone** link.
- The "star" icon allows you to save this zone information to your Home page.

Column descriptions

Column Name	Description / Actions
Name	The name supplied when this zone was created. Click on a Zone Name to see the Zone Details panel for that zone.
Descriptions	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Actions	<p>Delete - Use this link to delete the zone on the same row.</p> <p>Note: Deleting a zone does not delete resources in the zone. All agents (and their respective resources) move to the default zone if their current zone is deleted. The <i>default</i> zone cannot be deleted.</p>

Zone Details panel

This panel displays the zone name and description for the zone you selected.

Links and actions at the top of the table

- **Resources** - Use this link to go to the Resources page.
- **Edit** - Click this link to display the Edit Zone panel.

On the Edit Zone panel, you can change the zone name or add/change the zone Description. Click **OK** after making any changes.

- **Access Control** - Use this link to go to the Access Control page for this zone.
 - Current access privileges are displayed, if any.
 - Add the access control privileges you need for this zone.

Select the Properties tab to view any properties created for this zone, or use the "create" property links to create properties for this zone.

Creating a new zone

Fill-in the fields as follows:

Field Name	Description
Name	Supply a name of your choice for this zone. The name must be unique among other zone names.
Description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags.</p> <p>The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>.</p>

Click **OK** to see your new zone displayed in the table.

To add resources to this zone, go to the Resources page.

Access Control notes

A zone inherits privileges from the ZonesAndGateways ACL. See the [Access Control](#) ("Server Objects" section) Help topic for more information.

Resource and Resource Pool continue to inherit from Resources privileges. To create a resource, you must have modify privileges on Resources (this has not changed) and you must have modify privileges on the zone.

In addition, to move a resource from one zone to another, you must have modify privileges on both zones and the resource you want to move.

Gateways

To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the *source* zone and the other in the *target* zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).

If your company requires the added security of a firewall between zones, gateway agents can be configured to communicate with/through the firewall.

This page displays all gateways currently defined in ElectricCommander and provides other gateway operations—create, edit, delete, and so on.

- A firewall between zones - A gateway resource can be configured to communicate with an intermediary firewall in its path as a proxy to communicate with its peer on the other side of the gateway.
- Each gateway records the host/port combination each gateway agent/resource must use to communication with its peer on the other side of the gateway.
- Multiple gateways can be defined for a zone if required.
For example, you may have multiple resources in zoneA that need to communicate with each other, but some of those resources also need to communicate with zoneB, while others need to communicate with zoneC only. In this scenario, zoneA would require two gateways—one to zoneB and one to zoneC.
- One resource can participate in multiple gateways.
For example, assume we have 3 zones, zone1, zone2, and zone3, each created to contain agent/resource machines for a different, specific purpose (production, testing), but we want to share or pass data from a resource in zone1 to another resource in zone2 or zone3:
 - We need two gateways:
 - Gateway1 connects ResourceA in zone1 to ResourceC in zone3
 - Gateway2 connects ResourceA in zone1 to ResourceB in zone2
 - With this gateway-resource configuration, ResourceA can communicate directly with zone2 or zone3.

Links and actions at the top of the table

- To create a new gateway, click the **Create Gateway** link.
- The "star" icon allows you to save this gateway information to your Home page.

Column descriptions

Column Name	Description / Action
Name	The gateway name specified when this gateway was created. Click on a gateway name to see the Gateway Details panel for that gateway.

Column Name	Description / Action
Enabled	A "checkmark" in the box indicates this gateway is enabled.
Resource 1	The first of two resources required to create a gateway.
Host 1	The name Resource 2 uses to communicate with Resource 1. If "blank", the <i>Agent Host Name</i> attribute in Resource 1's definition is used at runtime.
Resource 2	The second of two resources required to create a gateway.
Host 2	The name Resource 1 uses to communicate with Resource 2. If "blank", the <i>Agent Host Name</i> attribute in Resource 2's definition is used at runtime.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Actions	Delete - Use this link to delete the gateway on the same row.

Gateway Details panel

This panel displays properties and access control privileges assigned to this gateway.

Select the Properties tab to see any existing properties or to create properties for this gateway.

Links and actions at the top of the panel

- **Edit** - Click this link to display the Edit Gateway panel.
On the Edit Gateway panel, you can change the gateway name or add/change the gateway Description.
- **Access Control** - Click this link to go to the Access Control page to set access privileges for this gateway.

Create Gateway panel

Supply the following information:

Field Name	Description / Action
Name	Supply a name of your choice for this gateway. The name must be unique among other gateway names.

Field Name	Description / Action
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
Enabled	Select this box to enable the gateway.
Resource 1	The name of your choice for the first of two required gateway resources. Do not include "spaces" in a resource name.
Host 1	The agent host name where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Leave this field blank to use the host name from <i>Resource1</i> 's definition.
Port 1	The port number used by <i>Resource1</i> - default is to the port number used by the resource.
Resource 2	The name of your choice for the second of two required gateway resources. Do not include "spaces" in a resource name.
Host 2	The agent host name where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Leave this field blank to use the host name from <i>Resource2</i> 's definition.
Port 2	The port number used by <i>Resource2</i> - default is to the port number used by the resource.

Click **OK** to see your new gateway displayed in the table.

Edit Gateway panel

This panel is populated with previously supplied information to define the gateway.

You can change any existing specifications or add new information.

Click **OK** to save your changes.

Access Control note

A gateway inherits privileges from the ZonesAndGateways ACL. See the [Access Control](#) ("Server Objects" section) Help topic for more information.

Define Search

The page allows you to search for any supported object type.

Fill-in the fields as follows:

Field Name	Description
Number of Results	Default is 200. Change this number to see a greater or lesser search result set.
Results Per Page	Choose the number of search results you would like to see. Default is 20.
Object Type	Use the drop-down menu to select the object you wish to search.
Intrinsic Filters	<p>Click the Add Intrinsic Filter link one or more times as needed to further define the sort criteria for your search.</p> <ul style="list-style-type: none"> Use the drop-down arrow to select an intrinsic filter argument. These arguments are properties. For a description of any of these properties, see the Properties help topic. Use the drop-down arrow to select an operator for the search. To narrow your search, supply an actual object name. For example, if you selected Project Name to begin your search, you might want to supply the actual project name in this field or the first few letters if you have multiple projects named in a similar pattern.
Custom Filters	Click the Add Custom Filter link one or more times as needed to enter custom sort criteria to further define your object search. Type-in the object name to define your search criteria, then select your search operator.

Click **OK** after filling-in the fields and to go to the Search Results page.

Notes:

- Only the first 450 characters of a property string are searchable. For example, if your property value happens to contain 1500 characters and the information you are trying to find is 800 characters into the property string, the search mechanism will not find it.
- When defining a search on this page, property path references are **not** allowed in search parameters. However, property path references are allowed in the Home page Quick View filters.

Search Results

This page displays the results of the search you defined.

- If the results were not as expected, click the **Edit Search** link to modify your search criteria.
- Click the **New Search** link to define a new object you need to find.
- Click the **Save Filter** link if you want to save this filter for future use in a report. In the pop-up screen, supply a name for the filter and click **OK**. The Saved Filter will be stored as a property in your chosen project.
- To continually monitor the status of your found object, click the **Refresh Search** link.

Notes:

If you need to refer to this page on a regular basis, mouse-over the "star" icon to add this page to the Shortcut section on your Home page.

If you use RSS, an active RSS icon is provided in Windows Explorer on this page for your convenience. If you use Firefox, click **Bookmarks** > Subscribe to this page to display the feed. You can then add the feed URL to a viewer of your choice.

Server

This page displays overall information about the ElectricCommander server.

Links at the top of the table

- Click the **Settings** link to go to the Edit Server Settings page.
Use this page to edit properties for the ElectricCommander server.
- Click the **Access Control** link to go to the Access Control page for the server.
See the [Access Control](#) Help topic for more information.

The tables...

- The System Access Control table displays top-level properties.
Click on an item in the Category column to see, add, or modify privileges for that object.
- In the Custom Server Properties table, click on a Property Name to view the property content.
If using *ectool*, these properties can be accessed using the property name starting with `"/server"`.

See the [Properties](#) Help topic for more information.

These links are provided within the table:

- **Create Property** - Use this link to create a new server property.
 - **Create Nested Property** - Use this link to create a new nested server property.
 - **Access Control** - This access control link allows you to create or edit privileges for the property sheet.
- The Actions column allows you to Edit or Delete the property in that row.

Settings - edit existing property settings

Use this page to edit property settings for the ElectricCommander server or any integrations you are using with Commander.

For example, if you access this page from the Server page, the page title will be Edit Server Settings and you will see fields for editing Commander server properties. From Perforce, the page will be titled Edit Perforce Settings, and the fields you can edit will be specific to Perforce functionality.

The following screen example is from the Edit Server Settings page.

Note: Depending on the Commander version you are using, the Edit Server Settings page may contain more or fewer fields than illustrated below.

For more information about any setting, hover your mouse over the setting name and click to see an "inline" help description about the field value.

hover your mouse over a field to see a Tool Tip.

Click **OK** when your changes are complete.

Also, notice the "star" icon in the upper right corner. Click this icon to add this page to your Home page for quick access if you need to change these settings frequently.

Source Control Configurations

This web page displays all source control configurations you have created to communicate with ElectricCommander.

[Link at the top of the table](#)

- Use the **Create Configuration** link to create a new or additional source control configuration.

[Table column definitions](#)

Configuration Name - This is the name you chose for this source control configuration.

Description - This is the plain text or HTML description you provided for this source control configuration.

Plugin - This is the Commander plugin associated with this source control configuration.

Actions -

Edit - Use this link to modify information previously supplied to configure this source control system.

Delete - Use this link to remove this source control configuration.

Source Control Configurations - create new or edit existing configuration

Use this page to define a new source control configuration (SCM) or modify an existing source control configuration. A configuration is a collection of properties that define how to communicate with a particular source control system.

ElectricCommander bundles and supports numerous source control types.

- After creating a source control configuration, your entry will appear in the table on the Source Control Configurations web page— to see this web page, select the Administration > Source Control tabs.
- Other SCM systems are bundled with Commander and available in the Plugin Catalog. Each integration contains options specific to that SCM.
- The following SCM integrations are provided as examples.

[AccuRev](#)

[ClearCase](#)

[File](#)

[Git](#)

[Perforce](#)

[Property](#)

[Subversion](#)

Note: If you need a different SCM, not specified in this help topic, go to the Plugin Manager page to locate your SCM (Administration > Plugins). See the Help topic associated with your plugin - located on the Plugins page. At a minimum, you will need to supply a name for your configuration and you may want a minimal text description.

AccuRev

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myAccuRevServer."

Description - You can change or modify the default text description - Commander does not use this information.

Login As

User Name - The name Commander needs to use to communicate with your AccuRev system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.

Password - The password for the specified User Name.

Retype Password - Type the password again.

Click **Submit** to save your information.

ClearCase

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myClearCaseServer."

Description - You can change or modify the default text description - Commander does not use this information.

Click **Submit** to save your information.

File

You may want to use this configuration with ElectricSentry to "watch" for changes in any files.

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration—any name you choose, but the name must be unique.

Description - You can change or modify the default text description - Commander does not use this information.

Click **Submit** to save your information.

Git

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration— any name you choose, but the name must be unique.

Description - You can change or modify the default text description - Commander does not use this information.

Click **Submit** to save your information.

Perforce

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "myPerforceServer."

Description - You can change or modify the default text description - Commander does not use this information.

Login As

User Name (P4USER) - The name Commander needs to use to communicate with your Perforce system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.

Password (P4PASSWORD) - The password for the specified User Name.

Retype Password - Type the password again.

P4PORT - Sets the hostname and port number ElectricSentry uses to contact the Perforce server (hostname:1234).

P4HOST - The name of the host machine to impersonate.

P4TICKETS - The full path name of the ticket file used by the "p4 login." This value is set as an environment variable, not as a command-line option (fullPathAndFileName)

P4CHARSET - Sets the character set used for translation of Unicode files (characterSet)

P4COMMANDCHARSET - Used to support UTF-16 and UTF-32 character sets (commandCharSet).

Click **Submit** to save your information.

Property

You may want to use this configuration with ElectricSentry to "watch" for any property changes.

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration—any name you choose, but the name must be unique.

Description - You can change or modify the default text description - Commander does not use this information.

Click **Submit** to save your information.

Subversion

Fill-in the fields as follows:

Configuration Name - Type a name for the configuration— any name you choose, but the name must be unique. For example, you might use "mySubversionServer."

Description - You can change or modify the default text description - Commander does not use this information.

Login As

User Name - This is the name Commander needs to use to communicate with your Subversion system. For example, you may be using a special "read-only" user name similar to "Build" for your user name.

Password - This is the password for the specified User Name.

Retype Password - Type the password again.

Click **Submit** to save your information.

Checking out code from a source control system

Go to Projects > select a project > select a procedure.

To create a New Step for source code management, select the **Plugin** link.

- In the Choose Step panel, select Source Code Management from the left pane, then select the SCM you configured.
- The right-pane now shows the types of steps available for your configuration. Select the step you need and automatically go to the New Step page.
- On the New Step page, notice the Subprocedure section now contains the SCM integration you configured and the step you chose.
- Supply the remaining information to create your SCM step.

To edit an existing source control configuration

Modify any of your source control configuration information by "typing-over" any previously entered information.

Click **Submit** to save your modified source control information.

The Home Page

[Job Configurations](#)

[Shortcuts](#)

[Jobs Quick View](#)

[Reports](#)

Overview

This page provides a convenient console for running jobs and viewing results.

- This web page is your Commander Dashboard for tracking project health.
- You can customize the page to display only the jobs that interest you and to provide one-click access to the jobs you run frequently. Each of the Home page sections can be customized.
- The Reports section allows you to see thumbnail report images that are updated each time the report is generated.
- This page provides shortcuts for quick access to pages you use most frequently in ElectricCommander or anywhere on the web.

IMPORTANT: If you are using or plan to use the EC-Homepage plugin to share your Home page sections, you will not see some links normally available to you. For example, if a particular Jobs Quick View category is shared, you **cannot** Delete or Modify that category. You can perform these functions only if the category belongs to you alone.

For information on sharing your Home page configuration, see the "[Home page configurations](#)" section in the "Customizing the Commander UI" help topic.

Job Configurations

Procedures in ElectricCommander can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem. When you create a job configuration, you supply all the information needed to run a procedure, including parameters and/or a credential. All job configurations are displayed here on the Home page. Invoke a particular configuration by clicking its name in the Job Configurations section.

Create Job Configurations three ways

- On the Home page, create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section.
 - In the Create Configuration popup menu, select the project and procedure you want to use for creating this configuration.

- From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page.
Your saved job configuration is displayed on your Home page.
- From the Edit Schedule page, click the **Save Configuration** link at the top of the page.
Your saved configuration is displayed on your Home page.

Shortcuts

Use shortcuts to save frequently visited Commander web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

Create Shortcuts two ways

- Mouse-over the "star" icon at the top of any ElectricCommander page and click "Add current page" to add that page to the shortcut list. Mouse-over the star icon again and click "Remove current page" to remove the shortcut for that page. The star icon is yellow for pages saved as a shortcut and gray for those pages not saved as a shortcut.
- Click the **Create** link in the Shortcut section and provide a name and URL to create a shortcut.

To modify or update a shortcut, click the **Edit** link adjacent to the shortcut you want to change.

Shortcuts can be accessed conveniently from any ElectricCommander web page. Mousing-over the star icon displays a list of shortcuts saved by the current user. Click on a shortcut name to view the page.

Note: The Shortcut section may contain static entries that cannot be deleted. And if you "share" your Home page, the Edit link will not be available for shared items.

Jobs Quick View

Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

The Jobs Quick View allows you to define job categories that are interesting to you.

The Home page displays the most recent jobs in each category, and you can easily click-through to get more details about any of those jobs. For example, clicking on a job name takes you to that job's Job Details page.

Create a job category

- Click the **Add Category** link in the Jobs Quick View section.
After creating a category, results are displayed on the Home page. Clicking the **Details** link displays a summary to the right of the category. In addition to job status and other diagnostic information, the summary displays running steps and failed steps (containing errors and warnings).
- Click the **Modify** link to edit the Jobs Quick View category or the **Delete** link to remove that job from the Jobs Quick View category.
- Click on the **Details** link to see job summary information—the summary remains visible, regardless of mouse location, until you click somewhere else on the page.

Note: If you "share" your Home page, the Modify and Delete links will **not** be available for shared items.

Reports

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

- Click the **Add Report** link to go to the New Reports page.
After filling-in the information on the New Reports page, you will see a thumbnail view of your report (after it runs) on your Home page.
Note: If the drop-down menu on this page is empty, click the **Help** link on the New Report page for more information.
- Click the Collapse/Expand (+/-) box to see the full thumbnail report or just the report title.
- Click the **Rename** link if you need to rename your report.
- Click the **Remove** link if you need to remove this report from your Home page.

Note: If you "share" your Home page, the Rename and Remove links will not be available for shared items.

The Home Page

[Job Configurations](#)

[Shortcuts](#)

[Jobs Quick View](#)

[Reports](#)

Overview

This page provides a convenient console for running jobs and viewing results.

- This web page is your Commander Dashboard for tracking project health.
- You can customize the page to display only the jobs that interest you and to provide one-click access to the jobs you run frequently. Each of the Home page sections can be customized.
- The Reports section allows you to see thumbnail report images that are updated each time the report is generated.
- This page provides shortcuts for quick access to pages you use most frequently in ElectricCommander or anywhere on the web.

IMPORTANT: If you are using or plan to use the EC-Homepage plugin to share your Home page sections, you will not see some links normally available to you. For example, if a particular Jobs Quick View category is shared, you **cannot** Delete or Modify that category. You can perform these functions only if the category belongs to you alone.

For information on sharing your Home page configuration, see the "[Home page configurations](#)" section in the "Customizing the Commander UI" help topic.

Job Configurations

Procedures in ElectricCommander can contain complex sets of parameters— making it difficult to remember the correct parameters for a particular situation and tedious to re-enter those parameters every time the procedure is invoked. Job Configurations provide a one-click solution to this problem. When you create a job configuration, you supply all the information needed to run a procedure, including parameters and/or a credential. All job configurations are displayed here on the Home page. Invoke a particular configuration by clicking its name in the Job Configurations section.

Create Job Configurations three ways

- On the Home page, create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section.
 - In the Create Configuration popup menu, select the project and procedure you want to use for creating this configuration.

- From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page.
Your saved job configuration is displayed on your Home page.
- From the Edit Schedule page, click the **Save Configuration** link at the top of the page.
Your saved configuration is displayed on your Home page.

Shortcuts

Use shortcuts to save frequently visited Commander web pages, so those pages are immediately accessible. You can create a shortcut to any page on the web also.

Create Shortcuts two ways

- Mouse-over the "star" icon at the top of any ElectricCommander page and click "Add current page" to add that page to the shortcut list. Mouse-over the star icon again and click "Remove current page" to remove the shortcut for that page. The star icon is yellow for pages saved as a shortcut and gray for those pages not saved as a shortcut.
- Click the **Create** link in the Shortcut section and provide a name and URL to create a shortcut.

To modify or update a shortcut, click the **Edit** link adjacent to the shortcut you want to change.

Shortcuts can be accessed conveniently from any ElectricCommander web page. Mousing-over the star icon displays a list of shortcuts saved by the current user. Click on a shortcut name to view the page.

Note: The Shortcut section may contain static entries that cannot be deleted. And if you "share" your Home page, the Edit link will not be available for shared items.

Jobs Quick View

Perhaps only a few jobs on this server are of interest to you. For example, you may care about jobs you launch manually and official builds for the products you work on, but you may not care about production builds for other products or personal jobs for other users.

The Jobs Quick View allows you to define job categories that are interesting to you.

The Home page displays the most recent jobs in each category, and you can easily click-through to get more details about any of those jobs. For example, clicking on a job name takes you to that job's Job Details page.

Create a job category

- Click the **Add Category** link in the Jobs Quick View section.
After creating a category, results are displayed on the Home page. Clicking the **Details** link displays a summary to the right of the category. In addition to job status and other diagnostic information, the summary displays running steps and failed steps (containing errors and warnings).
- Click the **Modify** link to edit the Jobs Quick View category or the **Delete** link to remove that job from the Jobs Quick View category.
- Click on the **Details** link to see job summary information—the summary remains visible, regardless of mouse location, until you click somewhere else on the page.

Note: If you "share" your Home page, the Modify and Delete links will **not** be available for shared items.

Reports

You can configure reports you would like to see on a regular basis and display a "thumbnail" report graphic in this section.

- Click the **Add Report** link to go to the New Reports page.
After filling-in the information on the New Reports page, you will see a thumbnail view of your report (after it runs) on your Home page.
Note: If the drop-down menu on this page is empty, click the **Help** link on the New Report page for more information.
- Click the Collapse/Expand (+/-) box to see the full thumbnail report or just the report title.
- Click the **Rename** link if you need to rename your report.
- Click the **Remove** link if you need to remove this report from your Home page.

Note: If you "share" your Home page, the Rename and Remove links will not be available for shared items.

Job Configuration

To create a new job configuration

Fill-in the fields as follows:

Name - Type a name you choose for the job configuration.

Procedure - Displays the current project and procedure. Click **Change** to use the popup to select a new project/procedure.

Parameters

Depending on the Project and Procedure you selected, a set of Parameters is displayed.

Fill-in all parameter values or just the required values if differentiated.

Advanced

Priority:

Use the drop-down menu to select the priority for this run procedure. Available priorities are: low, normal (default), high, or highest.

- You can select the job's priority here, on Run Procedure web page, or when you schedule the procedure to run at a regular time or interval. When a job is launched, its priority cannot be changed.
- 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.
- If using ectool, the priority can be set by passing "--priority=<low|normal|high|highest>" to a runProcedure, createSchedule, or modifySchedule operation.

Note: To raise the job priority level above "normal," the user who launches the procedure must have execute permission on the system-level access control list for priorities.

Impersonation:

Click one of the three "radio" buttons to determine the default login user to use when the configuration is run.

- Use pre-defined credential - If you select this option, ElectricCommander looks for a credential first in the top-level procedure and then in its project. If it finds a credential, it uses the login from that credential as the default for the job. If there is no credential in either place, by default, the job's steps run as the user under which the agent for the step is running.
- Use specific credential - If you select this option, you can choose a particular credential from those defined for the project containing the procedure - the login user from that credential is used as the default for the job.

- Use a specific user - If you select this option, you can enter a login name. When the configuration is run, you will be asked to supply the password for that account. The specified login will be used as the default for the job.

For more information on credentials and user impersonation, see the [Credentials and User Impersonation](#) Help topic.

Click **OK** when all selections are complete. Return to the Home page to see your new job configuration displayed.

To edit an existing job configuration

Change or modify any of your previously specified information.

Click **OK** when all selections are complete.

Return to the Home page to see your edited job configuration displayed.

Shortcuts

Use this page to create a **New Shortcut** or **Edit** an existing Shortcut.

Shortcuts appear on your Home page and provide quick access to other web pages, either in ElectricCommander or elsewhere.

- To create a new shortcut, type-in a shortcut name and the URL.
- To edit an existing shortcut, re-type the new shortcut name or URL (or both).

The easiest way to create a shortcut to an ElectricCommander page, such as the page for a particular project or procedure, is to navigate to that page and then click the "star" icon in the upper-right corner. This action automatically creates a shortcut and the star turns yellow to indicate a shortcut is in effect for the page.

If you click on the star icon again, the shortcut for that page is canceled.

Click **OK** to save your new or edited information and see your new/edited shortcut on the Home page.

Jobs Quick View

A quick view category provides a convenient way to select a group of jobs that interest you, such as "all production builds for product xyz version 3.6" or "all jobs I invoke manually." ElectricCommander displays the most recent jobs for each category on your Home page so you can review them easily.

To create a new jobs quick view category

Fill-in the fields as follows:

Name - Type a name for your new job category.

Number of Jobs - Type the maximum number of jobs you want to see in your quick view summary. The summary displays the most recent applicable jobs.

Include Last Success - "Check" this box if you want to see the last successful job in this category, even if it was not one of the most recent jobs.

Two Filter categories:

- Intrinsic filters: these filters provide a convenient way to access certain well-defined fields for jobs.
- Custom filters: these filters allow you to access a much broader range of values, including custom properties. Any values accessible through an intrinsic filter can be checked using a custom filter also (though not as conveniently).

Each filter specifies 3 values: the name of a particular property to check, a value it should be compared with, and relational operators such as "equals" or "less than." For example, if you select "Outcome" in an intrinsic filter with operator "not equals" and value "Success," the filter selects jobs that did not complete successfully because they had errors or warnings.

For an intrinsic filter, select a property by choosing one of several predefined properties from a list. For a custom filter, enter a property name. This name can be either the name of an intrinsic property for the job (such as "status" or "procedureName"), or the name of a property defined in the global property sheet for that job, or the name of a parameter for that job.

For the relational operator, select one of the values in the list. The behavior of the relational operators is determined by the underlying database. In most cases, comparisons are done using string comparison; operators such as "less than" may not be useful on numbers because "12" is considered less than "9". For some built-in properties, the database treats values according to a specific type such as number or date, so comparisons are implemented in the way appropriate for that type.

The relational operator "like" invokes an SQL wildcard comparison where "%" is the wildcard character. For example %test% matches "first test" and "tests failed" but not "Test 44".

Note: Job Quick View filters allow property path references. However, property path references are not allowed in search parameters on the Search page.

Click **OK** after making your selections to save your new job category and see it on the Home page.

To edit an existing jobs quick view category

Keep or change any of the previously entered information.

See the Filter information above to add or change existing custom or intrinsic filters.

Click **OK** after making your selections to save your modified job category and see it on the Home page.

New Report

Fill-in the following fields to define a new report to display on your Home page.

Field Name	Description
Project	The name of the project that contains the report you want to view.
Report	The report name you entered in the Report Title field when you configured this report. If you did not specify a "thumbnail" view when you defined your report, you will not see it here.
Title	This is the report title you want to see on your Home page to identify this report.

Click **OK** to continue.

After the report runs, the report you specified will be displayed on your Home page and updated whenever the report is generated again.

To populate the drop-down menu...

- Go to the Project Details page for the project containing the report (s) you want to view.
- Select the Report tab.
- Click the **Create Report** link.
- Check the Create thumbnail? checkbox, which enters this report's name in the New Reports drop-down menu.

Note: All reports, **except** the Procedure Usage report, are available as "thumbnail" reports on your Home page.

Active Users

This page displays all users known to this ElectricCommander server, including users defined locally within the server and those defined in external repositories such as LDAP or ActiveDirectory.

Active Users are those persons who have logged into Electric Commander, although if a another user views a selection of user accounts, those users [viewed] will appear on the Active Users page also.

Click the **Show Inactive Users** link to go to the All Users web page—this page lists all users: local, LDAP or Active Directory, active, or inactive. If you delete an LDAP or Active Directory user from this [Active Users] page, the user will not be deleted from the All Users page.

- To edit **local** user content, click on the user name you want to modify.
- To create a new local user, click the **Create Local User** link at the top, right-side of the table.
- To delete a local user, click the **Delete** link in the Action column for that user.
- Click the "star" icon at the top, right-side of the table if you want to add this page to your Shortcut section on the Home page.

While you can view **external** user content, you **must** go to the LDAP or ActiveDirectory repository to edit external user information. However, you may associate properties with external users and then use those properties in ElectricCommander.

To configure your existing LDAP and Active Directory account repositories to communicate with Commander, click the Administration > Directory Providers tabs.

Users and Groups

Use this page to view a filtered list of ElectricCommander local users or groups.

Fill-in the fields as follows:

Field Name	Description
Filter	Enter a partial name to retrieve the list of all users or groups that match that name.
Maximum Results	Supply the number of results you would like to see returned. Default is 100. Caution: Currently, ElectricCommander has no upward limit, but if you enter a number of 1000 or higher, you may overload your browser, which will result in performance degradation or worse.
Include Inactive [User/Group]	Use this checkbox only if requesting external LDAP or Active Directory providers. <ul style="list-style-type: none"> • If "checked," all available users or groups are returned. • If not checked, only those users or groups known to the Commander server are returned, for example, users who have logged in or groups containing users who have logged in.

Click **OK** to begin the search.

Links at the top of the page

Create Local [User/Group] - Click this link to create a new user or group - for users or groups *local* to the Commander server.

Column descriptions

Column Name	Description
Name	The name of the user or group. The name combined with the location is the unique identifier for this user or group, <i>local</i> to the Commander server. Click the name to see more information.
Repository	The repository where the user or group is located. "Local" means local to the Commander server. Other names in this column refer to defined directory providers.

Column Name	Description
Location	Only viewable for users or groups from non-local directory providers.
Real Name (users only)	The full name of the user.
Email (users only)	The email address for the user.
Actions	<p>Edit - Only available for local users and groups. Click to edit the user or group definition.</p> <p>Delete - Click to delete the user from the Commander server.</p> <p>Note: This delete function does not delete non-local users from an LDAP or Active Directory repository.</p>

User - create new or edit existing local user

Use this page to create or modify a *local* user **only**.

ElectricCommander supports two kinds of user accounts: those defined externally in an LDAP or Active Directory system, and *local* users defined in this ElectricCommander server.

Local users are not visible outside ElectricCommander.

Electric Cloud recommends using external accounts whenever available, but you may need to create local users if you do not have a shared directory service or if you need special accounts to use for Commander only.

To create a new local user

Fill-in the fields as follows:

Field Name	Description
User Name	Supply the name of the user account to be used for login
Real Name	Supply the user's real (given) name.
Email	Supply the user's email address.
Password	Supply the user's login password.

Click **OK** to save the new local user data.

To edit an existing local user

- You may highlight and type-over any previously entered user information.
- If you leave the password fields blank and click **OK**, the existing password remains unchanged.
- To change the user's password, enter the new password for the user, AND also enter the current admin password.

Click **OK** to save the modified user data.

Use this page to modify user properties or assign Custom User Properties. Select the **Create Property**, **Create Nested Sheet**, or **Access Control** links to set the properties you need.

Note: To configure your existing LDAP and Active Directory account repositories to communicate with ElectricCommander, click the Administration>Directory Providers tabs.

User Details

This page displays external user information retrieved from a repository such as LDAP or Active Directory.

To edit external user information, you **must** connect directly to the repository to modify external user content. However, you can associate properties with an external user and then use those properties in ElectricCommander.

Use this page to modify user properties or assign Custom User Properties. Select the **Create Property**, **Create Nested Sheet**, or **Access Control** links to set the properties you need.

Definitions for summary information at the top of page

User Name: The name this user is known by in the system.

Repository: Usually LDAP or Active Directory.

Real Name: The real, given name for user.

Email: The user's email address as known within the system.

Groups: This field displays all groups where this user is a member. Clicking on a group name displays the Group Details page, which lists all members who belong to that group.

Note: To configure your existing LDAP and Active Directory account repositories to communicate with Commander, click the Administration > Directory Providers tabs.

Edit User Settings

Use this page to modify your User Settings for the product tab view. Depending on which view you choose, you can have limited access to Commander features and functions.

In the Tab View, use the pull-down menu to make a new selection:

Default - Selecting this view provides all regular/standard product tab views, including plugins, which by default are installed into this view.

Base - This view removes the ability to use a plugin. You can see the plugin list (accessed by selecting the Plugins tab), but you are denied access to configure, install, uninstall, or promote a plugin.

Inherit - Frequently this view is set for a group. The UI searches for the property, `ec_ui/defaultView`, first on the user, second in any groups the user belongs to, and last, on the server. The inherit setting clears the user's private `defaultView` setting and then picks up the first value from the rest of the search path.

Click **OK** after selecting a different tab view.

Groups

This page displays all groups known to this ElectricCommander server, including groups defined locally within the server as well as those groups defined in external repositories such as LDAP.

- To edit a **local** group, click on its name.
- To create a new local group, click the **Create Local Group** link.
- Click on an **external** group to view its contents, but you cannot edit the content through ElectricCommander.
Instead, you must go to the group's repository directly. You can, however, associate properties with external groups, which can then be used in ElectricCommander.

To configure your existing LDAP and Active Directory account repositories to communicate with ElectricCommander, click the Administration>Directory Providers tabs.

Group - create new or edit existing local group

To create a new local group

Fill-in the fields as follows:

Name - Type a name for the group. Choose any name, but it must be unique among other local group names.

Users - Type a list of users who need to belong to this group— type one user per line.

Click **OK** after populating the group.

To edit a local group

- You can highlight the existing group name and type a new name—or
- You can scroll to the bottom of the list of users and type-in additional members for this group, one name per line—or
- You can highlight a user name and press the delete key (on your keyboard) to delete that user from the group.
- Click **OK** after completing your changes.

Also, you can associate custom properties with a group. Select either the **Create Property**, **Create Nested Sheet**, or the **Access Control** links to set up the properties you need.

Group Details

This page displays external group information retrieved from a repository such as LDAP or ActiveDirectory.

To edit external group information, you must connect directly to the repository to modify an external group. However, you can associate properties with an external group and then use those properties in ElectricCommander.

Click the **Access Control** link [at the top of the table] to see the privileges assigned to this group, inherited privileges, add a user or group, and more.

Also, you can associate custom properties with a group. Select either the **Create Property**, **Create Nested Sheet**, or the **Access Control** links to set up the properties you need.

Note: To configure your existing LDAP and Active Directory account repositories to communicate with ElectricCommander, click the Administration > Directory Providers tabs.

Directory Providers

ElectricCommander uses account information from multiple sources. In most cases, the primary account information source is an external LDAP or Active Directory repository: both user and group information is retrieved from the repository.

Local users and groups are defined within ElectricCommander.

Links at the top of the table

Click the **Add Active Directory Provider** or the **Add LDAP Provider** link to go to a new web page to configure your Directory Provider.

After configuring your directory provider, that name will appear in the All Providers table, the Provider Name column.

Column descriptions

Column Name	Description / Actions
Provider Name	Click on a Provider Name to make changes to an existing directory provider.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Actions	<p>Copy - Use this link to make a copy of the Provider Name configured on this row.</p> <p>Remove - Use this link to remove the Provider Name on this row.</p>

Detailed help for defining a directory provider in ElectricCommander is available from the [New Active Directory Provider](#) or the [New LDAP Provider](#) web page.

Click the **Help** link from either web page. The help text also contains examples you may find useful.

Directory Providers - create new or edit existing directory providers

Commander supports Active Directory and LDAP directory providers

This Help topic contains instructions for creating an Active Directory or LDAP directory provider and editing those configurations if you need to make changes at a later time. Use the following links to go the section you need:

[To create a new Active Directory provider](#)

[To create a new LDAP directory provider](#)

[Examples for directory provider field descriptions](#)

[To edit an existing directory provider](#)

To create a new Active Directory provider

Fill-in the fields as follows to specify your existing Active Directory users and groups to communicate with ElectricCommander. For examples of information you supply in the these fields, see the [table](#) after the following description sections.

Field Name	Description
<i>General section</i>	
Provider Name	This name identifies users and groups that come from this provider.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
URL Discovery	Select the method to retrieve the URL for the Active Directory server. Auto-discovery using DNS automatically discovers Active Directory servers on the given domain. Alternatively, you can specify a custom URL to an Active Directory server.
Domain Name	The domain where Active Directory server(s) are automatically discovered. For example, <code>my-company.com</code>

Field Name	Description
Use SSL	Select this box if you want to use SSL when the ElectricCommander server contacts your Active Directory server.
URL	This is an explicit URL to the Active Directory server. The URL is in the form <i>protocol://host:port/basedn</i> . Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but you can override the port if you cannot use the default location (default is port 389 for <code>ldap</code> or 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. Typically, this is the domain name where each part is listed with a <code>dc=</code> and separated by commas. Note: Spaces in the <code>basedn</code> must be URL encoded (use <code>%20</code> for spaces). If your site uses redundant directory servers for failover, you can supply multiple URLs separated by spaces.
Query User Name	The name of a user who has read-only access to user and group directories in Active Directory. This is the user name to use for fetching user and group information. When you provide a domain name, you can provide the simple name, for example, <code>myuser</code> . When you provide an explicit URL, you need to provide a distinguished name, for example: <code>cn=myuser,ou=Users,dc=electric-cloud,dc=com</code> .
Query User Password	The password for the query user.
<p>User Options section</p> <p>Note: When creating an Active Directory provider, the Commander server automatically sets default values for any options (fields) that are empty. The default values match the most common Active Directory configurations. After the provider is created, you can view and modify defaults by modifying the provider.</p>	
User Base	This string is prepended to the <code>basedn</code> to construct the directory DN containing user records.
User Search Filter	This LDAP query is performed in the context of the user directory to search for a user by account name. The string " <code>{0}</code> " is replaced with the user's login ID. Typically, the query compares a user record attribute to the substituted user login ID.
User Name Attribute	This is the attribute in a user record that contains the user's account name.
Full User Name Attribute	(optional) This is 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.

Field Name	Description
User Email Attribute	(optional) This is an attribute in a user record that contains the user's email address. If this attribute is not specified, the account name and domain name are concatenated to form an email address.
Search User Subtree	Select this checkbox to search the specified directory by the user base and all directories below. If this checkbox is not selected, the search is limited to the specified directory only.
<p>Group Options section</p> <p>Note: If you do not need group information at this time, do not fill-in the Group section—the "group test" will fail without this information, but the test will successfully authenticate users if both "user tests" are successful.</p> <p>When creating an Active Directory provider, the Commander server automatically sets default values for the options/fields that remain empty. These default values match the most common Active Directory configurations. After the provider is created, you can view and modify the defaults by modifying the provider.</p>	
Enable Groups	Select this checkbox to enable external groups for this directory provider.
Group Base	(optional) This string is prepended to the <code>basedn</code> to construct the directory DN containing group records.
Group Member Filter	(optional) This LDAP query is performed in the groups directory context to identify groups that contain a specific user. Two common forms of group records in LDAP directories are: <code>POSIX</code> style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Commander supports both forms, so the query is passed two parameters: " <code>{0}</code> " is replaced with the full user record DN, and " <code>{1}</code> " is replaced with the user's account name.
Group Member Attributes	(optional) This is a comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if a mixture of <code>POSIX</code> and <code>LDAP</code> style groups exist in the directory, multiple attributes might be required.
Group Search Filter	(optional) This LDAP query is performed in the context of the groups directory to enumerate group records. You can choose from common templates that include either security or distribution groups (or both). These templates are based on the most common Active Directory settings.

Field Name	Description
Unique Group Name Attribute	(optional) This is the group record attribute that contains the group name. To prevent group name overlap between multiple directory providers (or within the same provider in a multi-forested Active Directory server), Electric Cloud recommends setting this attribute to the <code>distinguishedName</code> .
Common Group Name Attribute	The Unique Group Name Attribute may not be searchable if using the <code>distinguishedName</code> . In this case, the Common Group Name Attribute can be used to search for groups, depending on the filter used.

After filling-in all fields, click the **Test** button.

Three tests validate the information you supplied:

- user authentication
- user identified in Active Directory
- find all groups where the user is a member

In case of test failure, correct the information you supplied and retest.

Click **Save** after successful test results.

New, defined directory providers will appear in the table on the Directory Provider web page.

To create a new LDAP directory provider

Fill-in the fields as follows to specify your existing LDAP users and groups to communicate with ElectricCommander. For examples of information you supply in the these fields, see the table after the following description sections.

Field Name	Description
General section	
Provider Name	This name identifies users and groups that come from this provider.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>

Field Name	Description
URL	The LDAP server URL is in the form <i>protocol://host:port/basedn</i> . Protocol is either <i>ldap</i> or <i>ldaps</i> (for secure LDAP). The port is implied by the protocol, but you can override the port if you cannot use the default (default port is 389 for <i>ldap</i> and 636 for <i>ldaps</i>). The <i>basedn</i> is the path to the top-level directory that contains users and groups at this site. Typically, this is the domain name where each part is listed with a <i>dc=</i> and separated by commas. Note: Spaces in the <i>basedn</i> must be URL encoded (use <i>%20</i> for spaces). If your site uses redundant directory servers for failover, you can supply multiple URLs separated by spaces.
Realm	This is the realm of the LDAP directory provider, which is used to create unique user names when you have multiple providers. For example, if the realm is <i>my-company.com</i> , users are saved as <i>myuser@my-company.com</i> .
Query User Name	This is the name of a user who has read-only access to the user and group directories in LDAP. This is the user name to use for fetching user and group information. When providing a domain name, you can provide the simple name, for example, <i>myuser</i> . When providing an explicit URL, you need to provide a distinguished name, for example: <i>cn=myuser,ou=Users,dc=electric-cloud,dc=com</i> .
Query User Password	This is the password for the query user.
User Options section	
User Base	This string is prepended to the <i>basedn</i> to construct the directory DN containing user records.
User Search Filter	This LDAP query is performed in the context of the user directory to search for a user by account name. The string " <i>{0}</i> " is replaced with the user's login ID. Typically, the query compares a user record attribute to the substituted user login ID.
User Name Attribute	This is the attribute in a user record that contains the user's account name.
Full User Name Attribute	(optional) This is 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.
User Email Attribute	(optional) This is 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.

Field Name	Description
Search User Subtree	Select this checkbox to search the specified directory by the user base and all directories below. If this checkbox is not selected, the search is limited to the specified directory only.
<p>Groups Options section</p> <p>Note: If you do not need group information at this time, do not fill-in the Group section— the "group test" will fail without this information, but the test will successfully authenticate users if both "user tests" are successful.</p>	
Enable Groups	Select this checkbox to enable groups.
Group Base	(optional) This string is prepended to the <code>basedn</code> to construct the directory DN containing group records.
Group Member Filter	(optional) This LDAP query is performed in the groups directory context to identify groups containing a specific user. Two common forms of group records in LDAP directories are: <code>POSIX</code> style groups where members are identified by account name, and <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Commander supports both forms, so the query is passed with two parameters: " <code>{0}</code> " is replaced with the full user record DN, and " <code>{1}</code> " is replaced with the user's account name.
Group Member Attributes	(optional) This is a comma-separated attribute name list identifying a group member. Most LDAP configurations only specify a single value, but if you have a mixture of <code>POSIX</code> and <code>LDAP</code> style groups in the directory, multiple attributes might be required.
Group Search Filter	(optional) This LDAP query is performed in the context of the groups directory to enumerate group records.
Unique Group Name Attribute	(optional) This is the group record attribute containing the group name.
Common Group Name Attribute	The Unique Group Name Attribute may not be searchable if using <code>distinguishedName</code> . In this case, the Common Group Name Attribute is used if searching for groups, depending on the filter.

After filling-in all fields, click the **Test** button.

Three tests validate the information you supplied:

- user authentication
- user identified in LDAP
- find all groups where the user is a member

In case of test failure, correct the information you supplied and retest.

Click **Save** after successful test results.

New, defined directory providers will appear in the table on the Directory Provider web page.

Examples for directory provider field descriptions

The following table provides examples for filling in the fields described above:

Field Name	LDAP example	ActiveDirectory example
Provider Type	LDAP	ActiveDirectory
Domain Name	N/A	my-company.com
Realm	my-company.com	N/A
URL	ldap://dir.company.com/dc=company,dc=com	ldaps://server/dc=company,dc=com
Query User Name	uid=JohnDoe,ou=People,dc=company,dc=com	cn=myuser,cn=Users,dc=company,dc=com
Query User Password	mypw	mypw
User Base	ou=People	cn=Users
User Search Filter	uid={0}	(&((userPrincipalName={0}) (sAMAccountName={0})) (objectClass=user))

Field Name	LDAP example	ActiveDirectory example
User Name Attribute	uid	userPrincipalName
Full User Name Attribute	gecos	name
User Email Attribute	mail	mail
Group Base	ou=Group	cn=Groups
Group Member Filter	((member={0}) (memberUid={1}))	member={0}
Group Member Attribute	member,memberUid	member
Group Search Filter	((objectClass=groupOfNames) (objectClass=posixGroup))	(objectClass=group)
Unique Group Name Attribute	distinguishedName	distinguishedName
Common Group Name Attribute	cn	cn

To edit an existing directory provider

You may change any previously supplied information in the fields.

After editing any fields, click the **Test** button.

The same three tests validate the information you supplied:

- user authentication
- user identified in the directory provider
- find all groups where the user is a member

In case of test failure, correct the information you supplied and retest.

Click **Save** after successful test results.

Edited, redefined directory providers will appear in the table on the Directory Provider web page.

Test Directory Provider

You must enter a valid User Name and Password on this page. Omitting or using an incorrect User Name and/or Password results in the test not completing or completing with an error. In either case, you will not obtain successful test results even if the Directory Provider information you supplied is valid and correctly entered into ElectricCommander.

Three tests validate the information you supplied:

- user authentication
- user identified in a directory provider
- find all groups where the user is a member

In case of test failure, correct the information you supplied and retest by clicking the **Test** button.

If you do not need group information at this time, and did not fill-in the Group section—the "group test" fails without this information, but the test will successfully authenticate users if both "user tests" are successful.

Workflows

This page displays all workflows in the ElectricCommander system, both running and completed workflows.

Links and actions at the top of the table

- Sort the Name, Project, State, Modify Time, Workflow Definition, and Completed columns by clicking on the column name.
After sorting a column, the page changes to a Workflow Search Results page and more search options are available.
- Define a search to see only the workflows you choose. Click the **New Search** link.
- The "star" icon allows you to save this web page to your Home page.

Column descriptions

Name - This is the Commander-generated workflow name, which is defined by the Workflow Name Template setting. Click on any workflow name to go to that workflow's Workflow Details page.

Project - The name of the project containing this workflow. Click on any project name to go to the Project Details page for that project.

State - The current state of the workflow.

Modify Time - The time this workflow was modified.

Workflow Definition - The name of the workflow definition that ran to create this workflow. Click on any workflow definition name to go to that workflow definition's Workflow Definition Details page.

Completed - A "checkmark" in this column indicates the workflow has completed.

Actions - **Delete** - Use this link to delete the workflow on the same row.

Workflow Definition - create new or edit existing workflow definition

To create a new workflow definition

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique workflow definition name of your choice. You may want your workflow definition name to reflect the project where it belongs or its purpose. For example, you might set a name based on a product or team using this workflow.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Workflow Name Template	<p>This is the template used to determine the default name of jobs launched from the workflow definition.</p> <p>For example:</p> <pre> \$[projectName]_\${[/increment /myproject/workflowCounter]}_ \$[/timestamp] </pre> <p>(substitute your values for the names above)</p> <p>Produces a workflow name like:</p> <pre> projectFoo_123_20140102130321 </pre> <p>Supply any combination of elements to create workflow names more meaningful to you. For example, you could choose to include the build number.</p> <p>Note: Electric Cloud does not recommend using the Commander-generated <code>workflowId</code> because it is no longer a human-readable integer so it does not provide any identifiable information and cannot be used as a counter.</p>

Click **OK** to save the information you entered.

Your new workflow definition will appear on the Project Details > Workflow Definitions page.

To edit an existing workflow definition

Want to rename the workflow definition? Type a new name in the Name field and click **OK**.

You can change or add to the description or modify the template.

Click **OK** when your edits are complete.

Workflow Definition Details

This help topic provides a detailed overview for all views and functionality on this web page. You may want to read this topic in its entirety if this is your first experience with the Commander Workflow feature. If you are an experienced workflow user, the following quick links will take you to a specific topic section for quick review.

A note to existing workflow users: We recommend reviewing the Graph view section because workflow functionality previously found on other Commander web pages is now combined for ease-of-use on the Graph page.

Graph view

[To create a New State](#)

[To create a State Definition](#)

[Action](#)

[Notifiers](#)

[Parameters](#)

[Properties](#)

[To create a Transition Definition](#)

[Using the graph's quick-access features](#)

[Show Legend](#)

List view

Properties view

This page displays workflow definition components, including state and transition definition names, types, actions, descriptions, and various links to create additional transitions, state definitions, define access control, run the workflow, and more. You can view any previously defined workflow objects and create new properties too.

Opening to the Graph view, your first glance at this page will look similar to the following:



Links and actions at the top of the page

- Above the Workflow Definition Details page title you see (in this example) "Project: Learning Workflows". This breadcrumb information tells you which project you are viewing or working with and provides a link back to the project, itself.
- Immediately after the page title, you see "Commander release", which (for our example) is the name of this workflow.
- **Run** - Use this link to run the workflow definition. Hover your mouse over the drop-down arrow to see these choices:
 - Selecting **Run...** - allows you to pick the "starting state" to run the workflow. After clicking **OK** from the Run Workflow popup dialog, the Run Workflow page is displayed. If you previously created parameters for this workflow, they are displayed. You can accept the parameter values or change them before running the workflow.
 - Selecting **Run Immediately** - this option uses the first (default) "starting state" to run the workflow.
Note: Clicking Run, without using the down-arrow to make a selection, is the same as selecting Run Immediately.
- **Edit** - Use this link to go to the Edit Workflow Definition page to make any necessary changes.
- **Access Control** - Use this link to set privileges for this workflow definition. ***For more information,*** see the Access Control Help topic.
- The "star" icon allows you to save this workflow definition to your Home page.

Navigation and view summary

Each button has its own expanded description section following this summary.

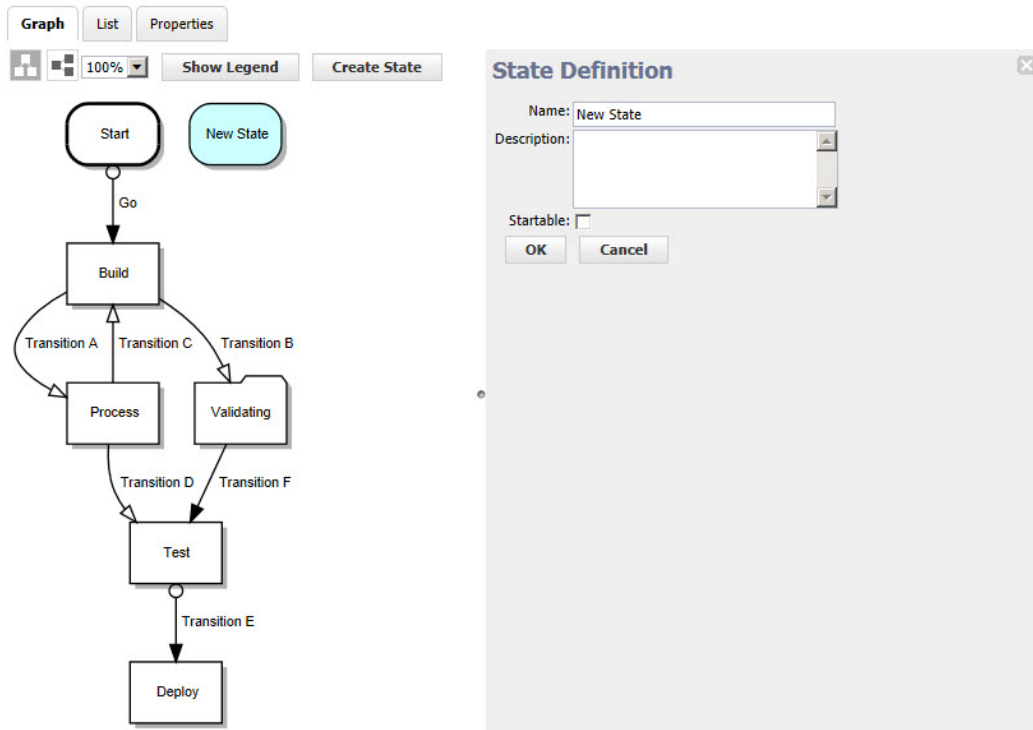
- **Graph** - the page opens in this view. The main portion of this view contains the work area for visually creating your workflow.
- **List** - this view provides a table displaying states, transitions, and more in list form.
- **Properties** - this view provides a table displaying previously created properties and provides links to create additional properties.
- **Show Legend** - displays a legend for workflow objects.
- **Create State** - this button produces the State Definition panel where you can create new states as you build or modify your workflow.
- The multi-square directional icons allow you to change your workflow graph from vertical to horizontal, depending on how you choose to see the graph.
- The drop-down menu for percentages allows you to diminish or expand the workflow graph size.

Graph view

To create a workflow, minimally you need to create state definitions and transition definitions. Optimally, those objects will have parameters and properties, and you will want to create or enforce access control and perhaps create one or more email notifiers too. To begin, the following sections describing functionality in this view will familiarize you with how to create workflow objects and end with exploring the graph's quick-access features.

Click the **Create State** button to create a new state using the State Definition panel.

In the following screen example, you see a New State object added next to the Start state in an existing workflow graph.



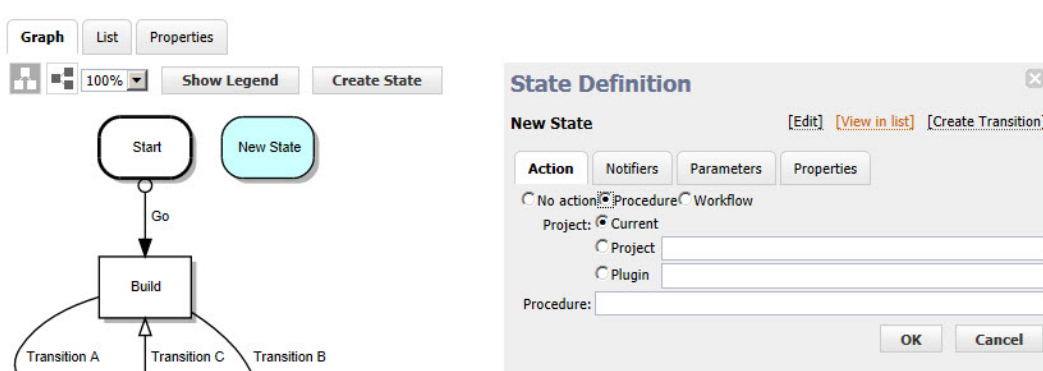
To create a new state

After clicking the Create State button, fill-in the State Definition panel as follows:

Field Name	Description
Name	(Required) Provide any name of your choice. The name must be unique to this workflow definition. If you do not provide a name, a system-generated name is created: New State, New State 2, and so on.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
Startable	Select this checkbox if you want this state to be your workflow "starting" state. The first state in any workflow definition is saved as <i>Startable</i> , but you can change this later.

Click **OK** and the State Definition panel provides options to configure your new State Definition.

To create a new State Definition



The top of the State Definition panel displays the state name. In this example, "New State" is the state name.

Links at the top of the panel

- **Access Control** - use this link to add or modify access control privileges for this state (link not illustrated in the State Definition panel example above).
- **Edit** - use this link to modify this state's name, description, or whether or not the state is *Startable*.
- **View in list** - displays the List view to see the state definition in the Workflow Definition Details table. This state will be highlighted in the List view.
- **Create Transition** - displays the Transition Definition panel to create a transition for this state.

The "tabbed" or button sections

Action

As each state in a workflow becomes active, it performs an *action*. The state's action can be to create a job from a procedure or to start a workflow from a workflow definition. When the job or called workflow completes, On Completion transitions are evaluated to see if the workflow should change to a different state, possibly based on the outcome of the action.

Selecting *No action*, *Procedure*, or *Workflow* changes the fields that appear under these choices.

- **No action** - if selected, no other information is required. Click **OK**.
- **Procedure** - this selection displays the following fields:
 - **Current** - this selection refers to the project that contains the workflow definition.
 - **Project** - if selected to call a different project, start typing in the text box to display a list of (non-plugin) projects from which to select. Only the first 10 matches are displayed. Type more characters to refine the list.

- Plugin - if selected, start typing in the text box to display a list of plugin projects from which to choose. Only the first 10 matches are displayed. Type more characters to refine the list.
- Procedure - start typing in this field to see a list of available procedures from which to select, depending on your previous choice of Current, Project, or Plugin. Only the first 10 matches are displayed. Type more characters to refine the list.
- Workflow - this selection displays the following fields:
 - Similar to selecting Procedure (above), select Current, Project, or Plugin.
 - Workflow - if selected to call another workflow definition, start typing in this text box to display a list of workflow definitions from which to choose. Only the first 10 matches are displayed. Type more characters to refine the list.
 - Starting State - start typing in this field to see a list of available starting states definitions in the workflow definition you selected. Only the first 10 matches are displayed. Type more characters to refine the list.

After selecting the procedure or starting state, press the <Tab> key to leave the field and see a list of parameters for the selected action. After entering any procedure values, click **OK** to save the changes.

Notifiers

Select this tab to see a list of email notifiers previously created for this state definition or to create an email notifier. Notifiers are commonly used to inform interested parties about transitions to the state being defined.

Note: Before you can set up an email notifier, you need an email configuration. If you have not already defined an email configuration, you can do it now. Go to Administration > Email Configurations and select the **Add Configuration** link.

The Notifiers table contains a list of all previous created notifiers for this state definition.

Column descriptions

Column Name	Description / Actions
Name	The name of the email notifier. Select a name to go to the "edit" page for that notifier if you need to make changes.
Type	The type of email notifier specified during creation.
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> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
Actions	Delete - Use this link to delete the notifier on that row.



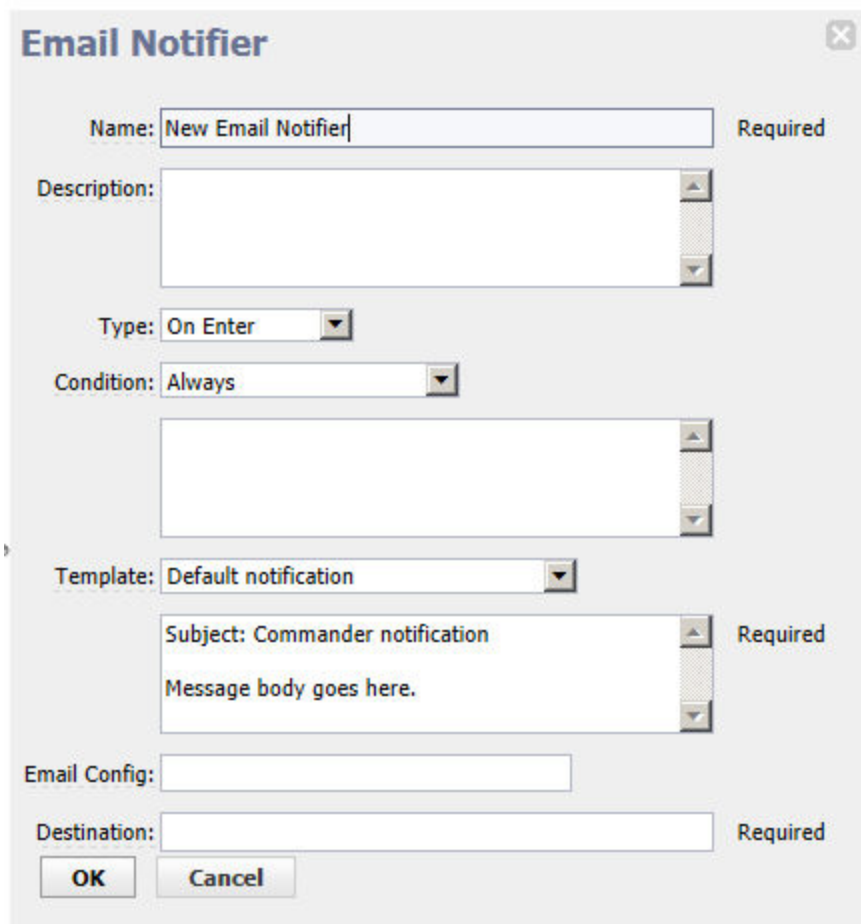
State Definition [X]

New State [Edit] [View in list] [Create Transition]

Action **Notifiers** Parameters Properties

Name	Type	Description	Actions
New Email Notifier	onEnter		Create Notifier Delete

To create an email notifier, click the **Create Notifier** link to access the Email Notifier panel.



Email Notifier [X]

Name: Required

Description:

Type:

Condition:

Template:

Subject: Required

Message body goes here.

Email Config:

Destination: Required

Fill-in the fields as follows:

Field Name	Description
Name	(Required) This name can be any text string you choose. The name must be unique among other notifier names in this project, on procedures or workflow definitions called from other projects.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Type	<p>Use the pull-down menu to choose a type:</p> <ul style="list-style-type: none"> On Completion Notifier - sends an email after the state's job or workflow completes. If no job or workflow is defined for that state, these notifiers will not be sent. On Enter Notifier - sends an email when the state becomes the workflow's active state. On Start Notifier - sends an email after the state's job or workflow starts. If no job or workflow is defined for that state, these notifiers will not be sent.
Condition	Use the pull-down menu to select the type of condition you need for this email notifier. Edit the auto-supplied condition in the text box or add a completely new script for your purpose. The condition specifies whether the notifier should send a message depending on the result of a property expansion. If the result is empty or non-zero, the message is sent. If the result is "0", the message is not sent.
Template	<p>Use the pull-down menu to select from a list of global, ready-to-use formatting templates. Depending on the type of email notifier you are creating, the available template choices in the drop-down menu will be different.</p> <p>(Required) To customize your template, edit the auto-supplied text in the text box or you can add a completely new script for your purpose. Any edits made in this text box will not be saved to the global template.</p> <p>To create a custom template, the basic structure is:</p> <ul style="list-style-type: none"> zero or more email header lines blank line message body

Field Name	Description
Email Configuration	Click inside this field or start typing to bring up a list of possible email configuration names. An email notifier that does not specify an email configuration will use the configuration named 'default' if it exists.
Destinations	(Required) This is a space-separated list of valid email addresses, email aliases, or ElectricCommander user or group names, or a property reference that expands into such a list, or you can supply an LDAP DL name (group name).

Click **OK** to save your email notifier configuration. The next time you view the Notifier table, you will see this notifier in the list.

Parameters

Similar to Notifiers, a parameter table is displayed containing all formal parameters defined for this state definition. Parameters are presented when launching the workflow definition (if it is startable), or when taking a manual transition to the state. A transition may supply values for some or all of the formal parameters defined by the state definition, in which case only the unmapped parameters are presented to the user.

- **Create Parameter** - Use this link to go to the New Parameter page to create a parameter for this state definition.

If you need assistance with creating a parameter, click the **Help** link in the upper-right corner of the New Parameter page.

Properties

Similar to the Notifiers and Parameters table, a properties table is displayed containing available properties for this state definition. If no properties were defined for this state definition, the table does not exist.

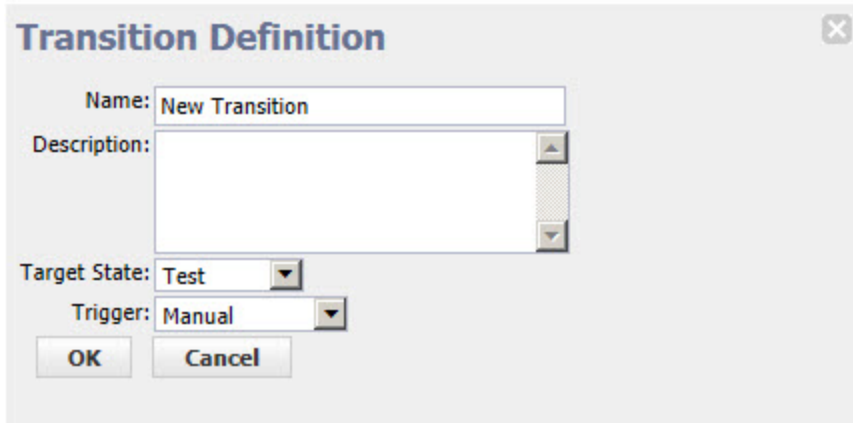
Click one of the available links to create one or more properties that will be displayed in the table when created:

- **Create Property** - Click this link to go to the New Property pop-up box to create a new property for this state definition.
- **Create Nested Sheet** - Click this link to go the New Nested Property Sheet pop-up box to create a nested property.
- **Access Control** - Click this link to go to the Access Control page for the property sheet.

The property pop-up boxes contain a Help link if you need assistance creating a property.

To create a Transition Definition

On the State Definitions panel, click the **Create Transition** link to see the Transition Definition panel:



The image shows a 'Transition Definition' dialog box. It has a title bar with a close button (X). Inside, there are four labeled fields: 'Name' with the text 'New Transition', 'Description' with an empty text area and vertical scrollbars, 'Target State' with a dropdown menu showing 'Test', and 'Trigger' with a dropdown menu showing 'Manual'. At the bottom are 'OK' and 'Cancel' buttons.

Fill-in the fields as follows:

Field Name	Description
Name	(Required) Supply a unique name for the transition definition—any name you choose, but the name must be unique within the state definition. If you do not provide a name, a system-generated name is created: <i>New Transition</i> , <i>New Transition 2</i> , and so on.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Target State	This field displays the "best guess" for the target state. If not correct, or you intended to create a transition for a different state, use the down-arrow to choose an available target state from the list.

Field Name	Description
Trigger	<p>Use the drop-down menu to choose the transition type.</p> <p>The four types of transitions are:</p> <ul style="list-style-type: none"> On Completion - transition occurs when the action completes. These transitions are ignored if no action is specified for the source state. Note: On Completion transitions are taken only if the state is still active when the action completes, and are ignored if the workflow has transitioned to another state—this can occur if an On Start or Manual transition occurred before the action completed. On Enter - transition occurs before sending notifiers or starting the action. On Start - transition occurs immediately after starting the action. These transitions are ignored if no action is specified for the source state. Manual - transition occurs when a user selects the transition in the UI and specifies parameters. The same action can occur using ectool or the Perl API by calling <code>transitionWorkflow</code>. Only users who have "execute" permission on the transition are allowed to use a Manual transition.
Condition	<p>Use the drop-down menu to choose a condition. After selection, the text box is populated with a sample JavaScript string to edit for your purposes. Note: If you select Custom, no sample is provided—supply a script in the text box. Using JavaScript is recommended.</p>

Click **OK** to save your transition definition.

Notice the new links and options available now:

- **Access Control** - use this link to add or modify access control privileges to this transition definition.
- **Edit** - use this link to modify the transition definition.
- **View in list** - use this link to see the new transition definition in the List view (table) on the Workflow Definition Details page. You can use the List view to move the transition to a different position within the state definition.
- **Parameters** - if parameters were defined on the State Definition Details page, you will not see them here. If parameters are defined on this page, you need to supply values. Manual transitions allow you to defer parameter assignment until transition time.
- **Properties** - displays properties available for this object or you can create one or more properties at this time.

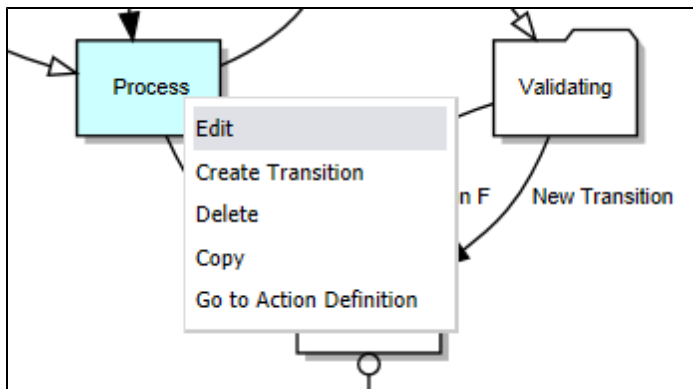
Using the graph's "quick-access" features

The following screen example is a portion of a workflow graph.

In the graph, states and transitions are links:

- Click on a state to open the State Definition panel to modify that state.
- Click on a transition to open the Transition Definition panel to modify that transition.
(Hover your mouse over a transition or transition name and click when it changes color.)

However, if you **right-click** on a state or transition, a context-sensitive menu with other options is available. In the following example, right-clicking on the "Process" state provided the quick-link pop-up menu.



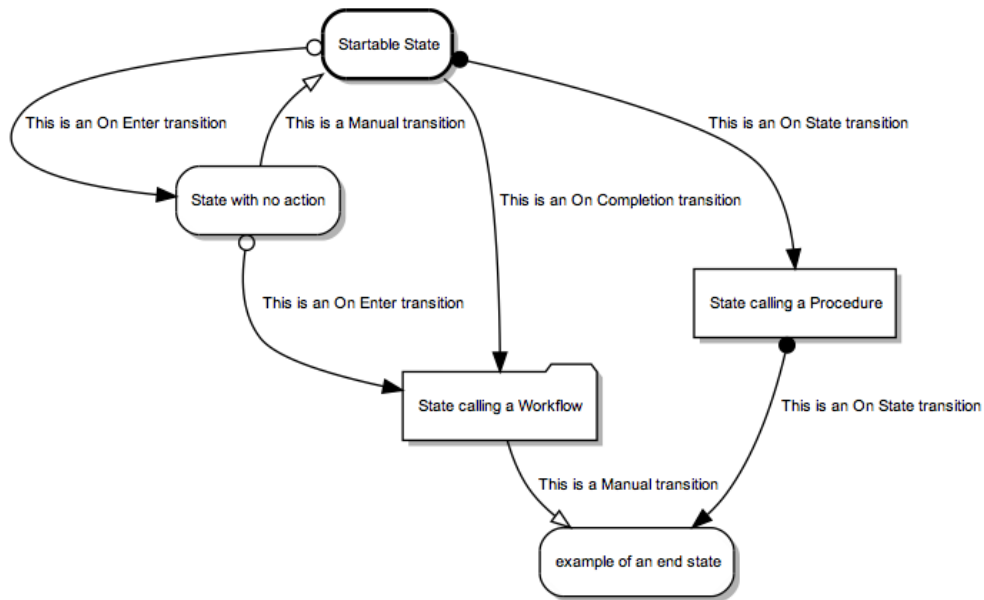
Available pop-up links include:

- For states calling a procedure or another workflow, menu choices can be:
 - Edit - opens the State Definition panel to make modifications. This is the same as selecting the Edit link on the State Definition panel.
 - Create Transition - opens the Transition Definition panel.
 - Delete - deletes the state.
 - Copy - makes a copy of the state with all associated transitions preserved.
 - Go to Action Definition - for states calling a procedure, the Procedure Details page is opened. For states calling a workflow, the Workflow Definitions Details page for the "subworkflow" is opened.
- For states with "No action", the choices are the same, except the "Go to Action Definition" option is not available.
- For transitions, menu options include: Rename, Delete, and Copy.
- Right-clicking anywhere on the graph canvas provides two more choices:
 - Create State - allows you to quickly create a new state. This is the same as clicking the Create State button at the top of the graph.
 - Rotate Graph - acts as a "toggle", rotating the graph from a vertical to horizontal view and back again.

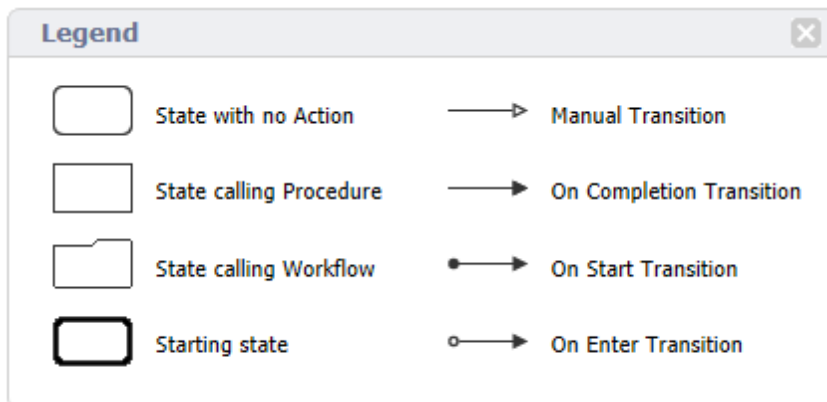
Show Legend

The following screen is another example of a workflow graph. Depending on your workflow, your graph could be very simple or much more complex than the example below.

Notice the transition line-endings and different shapes used for states.



The legend is available for reference anytime to help you become familiar with state shapes and line-ending definitions.



List view

This view will be empty until you build a workflow in the Graph view. You can view this page at any time to see a list of states and transitions in your workflow definition.

The following screen is an example of a populated Workflow Definition Details page. Notice the **Run**, **Edit**, and **Access Control** links are still available at the top of the page. A **Create State Definition** link is available above the Action column, which has the same function as clicking the Create State button in the Graph view.

Workflow Definition Details – Commander-release

 Run  Edit  Access Control 

Graph List Properties Create State Definition					
Name	Type	Action	Startable	Description	Actions
Start	No action		✓	this is the starting state	Create Transition Copy Move Delete
Go	On Enter	Target: Build			Access Control Copy Move Delete
Build	Subjob	Learning Workflows : Build			Create Transition Copy Move Delete
Transition A	Manual	Target: Process			Access Control Copy Move Delete
Transition B	Manual	Target: Validating			Access Control Copy Move Delete
Process	Subjob	Learning Workflows : Process			Create Transition Copy Move Delete
Transition C	Manual	Target: Build			Access Control Copy Move Delete
Transition D	Manual	Target: Test			Access Control Copy Move Delete
Validating	Subworkflow	EC-Examples : Check Prerequisites			Create Transition Copy Move Delete
Transition F	On Completion	Target: Test			Access Control Copy Move Delete

Column descriptions

Column Name	Description / Actions
Name	<p>This column displays state definition and transition definition names currently defined for this workflow definition.</p> <ul style="list-style-type: none"> Transition definition names are indented under a state definition name. Clicking a state definition name displays the State Definition panel (in the Graph view) for that state to see how it was defined or to make modifications. Clicking a transition definition name displays the Transition Definition panel (in the Graph view) for that transition to see how it was defined or to make modifications.
Type	<p>This is the type of action or trigger currently assigned to the state definition or transition definition.</p> <ul style="list-style-type: none"> For states - the type may show calling a subjob, subworkflow, or no action. For transitions - the trigger type is provided.

Column Name	Description / Actions
Action	<p>This is the action previously defined for the state or transition.</p> <ul style="list-style-type: none"> For states - (where the Type is "subjob") the format for actions is <code><projectName>:<procedureName></code>. Clicking the first link displays the Project Details page for the workflow project. Clicking the second link displays the Procedure Details page for the procedure used by that state. For states - (where the Type is subworkflow) the format for actions is <code><projectName>:<workflowName></code>. Clicking the first link displays the Project Details page for the workflow project. Clicking the second link displays the Workflow Definition Details page for the workflow used by that state. For transitions - clicking the Target action displays the State Definitions panel in the Graph view.
Startable	A "checkmark" in this column shows this state was defined as startable, which means this state is the beginning point for your workflow.
Description	A plain text or HTML description for this object.
Actions	<p>Create Transition - Click this link to go to the Transition Definition panel to create a new transition definition for this state.</p> <p>Access Control - Click this link to go to the Access Control page to define access privileges for this transition definition.</p> <p>Copy - Use this link to make a copy of either the state definition or the transition definition. Note: Copying a state also copies its transitions, however, you must have modify privileges on the workflow project to make a copy.</p> <p>Move - Use this link to move a state to another position in the workflow. You will be prompted to choose a state name to move this state to a location before that state. If moving a transition, note that transition can be moved only to another location for the same state—transitions cannot be moved from one state to another state.</p> <p>Delete - Use this link to delete the state definition or transition definition on the same row.</p>

Properties view

Select the Properties view button to see the Properties table. The following screen is an example of the Properties table. If no properties were defined for this workflow definition, this table will be blank.

Graph

List

Properties

Create Property

Create Nested Sheet

Access Control

Property Name	Value	Description	Actions
prop1	build_1_1	beta release	Delete
prop2	build_1_2	beta2release	Delete

The following links and actions are available in the Properties table:

Links and actions at the top of the table

- **Create Property** - Click this link to go to the New Property pop-up box to create a new property for this workflow definition.
- **Create Nested Sheet** - Click this link to go to the New Nested Property Sheet pop-up box to create a nested property.
- **Access Control** - Click this link to change access control privileges on the property sheet.

Column descriptions

Column Name	Description / Actions
Property Name	Select a property name to edit that property. You can change its name, value, or add/change its description. If the property name is preceded by a folder icon, this is a property sheet. Click on the property name to open the "folder".
Value	The value currently assigned to the property.
Description	A plain text or HTML description for this object.
Actions	Delete - Click this link to delete the property on that row.

Run Workflow

Use this page to run a workflow.

- The "star" icon allows you to save this job information to your Home page.
- The "bread crumbs" Project: SoftwareBuild / Workflow: template-99 provide links to a previous web page.
- The name after the Run Workflow page title is the name of the workflow you intend to run.

Project: SoftwareBuild / Workflow Definition: workflow-template

Run Workflow — workflow-template



Starting State: building

Parameters

param1:

Required

param2:

param3:

Required

OK

Cancel

Starting State - This is the name of the current starting state for this workflow. If this workflow has multiple starting states and this is not the one you want to use, return to the Run Workflow pop-up dialog and select a different starting state.

Parameters -

- Any parameters previously specified for this starting start are displayed. If no parameters were defined, this area will be blank.
- If values are supplied, these are the default values specified when the parameter was created. If necessary, you can "type-over" these values to change them before running this workflow.
- You must supply a value for any blank value field labeled "Required".

Click **OK** to run the workflow when your selections are complete.

Workflow Details

This page displays workflow details, including the workflow state, project name, workflow definition name, and any properties. The following links are provided for quick access to topic sections you may want to review again.

[Graph](#)

[Show Legend](#)

[Using the graph's "quick-access" features](#)

[Show History](#)

[States](#)

[State Details panel](#)

[Parameters](#)

[Properties](#)

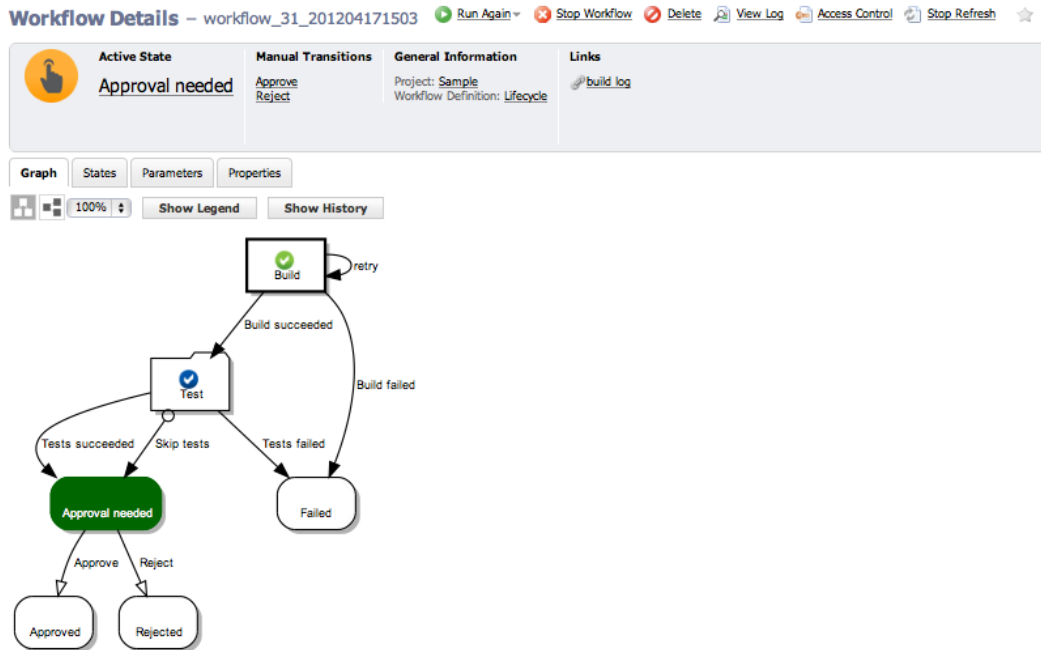
The name after the Workflow Details page title is the name of the workflow created after running a Workflow Definition. The number at the end of the workflow name is the workflow definition ID number. The ID number is auto-generated by Commander. Your workflow name will be different, depending on the information you supplied in the workflow definition.

Links and actions at the top of the table

Additional links are available if the workflow is running. When the workflow is complete, some links are no longer available.

- **Run Again** - Use this link to run the workflow definition again.
When you select the **Run Again** button, you are requesting to re-run the workflow definition with the same parameter values as the original invocation.
If you want to change the parameter values, you can select "**Run...**" from the drop-down menu (small down-arrow) next to the Run Again button.
- **Stop Workflow** - Select this option to stop the workflow. Any unfinished processes will complete, but no new transitions will occur.
- **Delete** - Deletes this workflow.
- **View Log** - Use this link to view the log created by this workflow.
- **Access Control** - Use this link to set privileges for this workflow. For more information, see the [Access Control](#) Help topic.
- **Stop Refresh** - This link is available only if a workflow is currently running. Click this link if you do not want the page to refresh the page automatically while the workflow is running.
- The "star" icon allows you to save this workflow information to your Home page.

While the workflow is running, your Workflow Details page will be similar to the following example:



Summary section - at the top of a running workflow page

- Next to the icon, notice that the state name, "Approval needed", is a link. Click this link to open the State Details panel for this state.
- The **Manual Transitions** section allows you select a manual transition if any were defined. However, if the manual transition contains any parameters whose values were not supplied at definition time, selecting that transition takes you to the Transition Workflow page to supply values.
- The **General Information** section provides links from the Project name to the corresponding Project Details page and from the Workflow Definition name to its corresponding Workflow Definition Details page.

Links

This page can include a Links section also, which will appear on the far right-side of the Summary section.

To add a link to this section:

- **Create a link to any file:**

To add a link, run a command in any job's job step using this format (this works for both local [disconnected] and non-local workspaces):

```
ectool setProperty "/myJobStep/report-
urls/<
myReportName>"/commander/jobSteps/<jobStepId>/<artifactDirectoryName>/<file-
path>
```

Example command:

```
ectool setProperty "/myJobStep/report-urls/Test Report"/commander/jobSteps/
$[/myJobStep/jobStepId]/testreport/index.html
```

Note: If you are using this format (from Commander versions prior to v4.2):

```
ectool setProperty "/myJob/report-urls/<myReportName>"/commander/jobs/<jobId>/<workspaceName>/<artifactDirectoryName>]/<file-path>
```

Example command:

```
ectool setProperty "/myJob/report-urls/Test Report"/commander/jobs/${myJob/jobId}/  
${myJobStep/workspaceName}/testreport/index.html
```

it will continue to work for non-local workspaces only. It is recommended, however, to change the command to use the newer format.

- **Create a link to any directory:**

To add a link, run the following style command in any job's job step:

```
ectool setProperty "/myJobStep/report-urls/Main Job Workspace"  
"file:///WinStor2/scratch/chron55build/${myJob/jobName} "
```

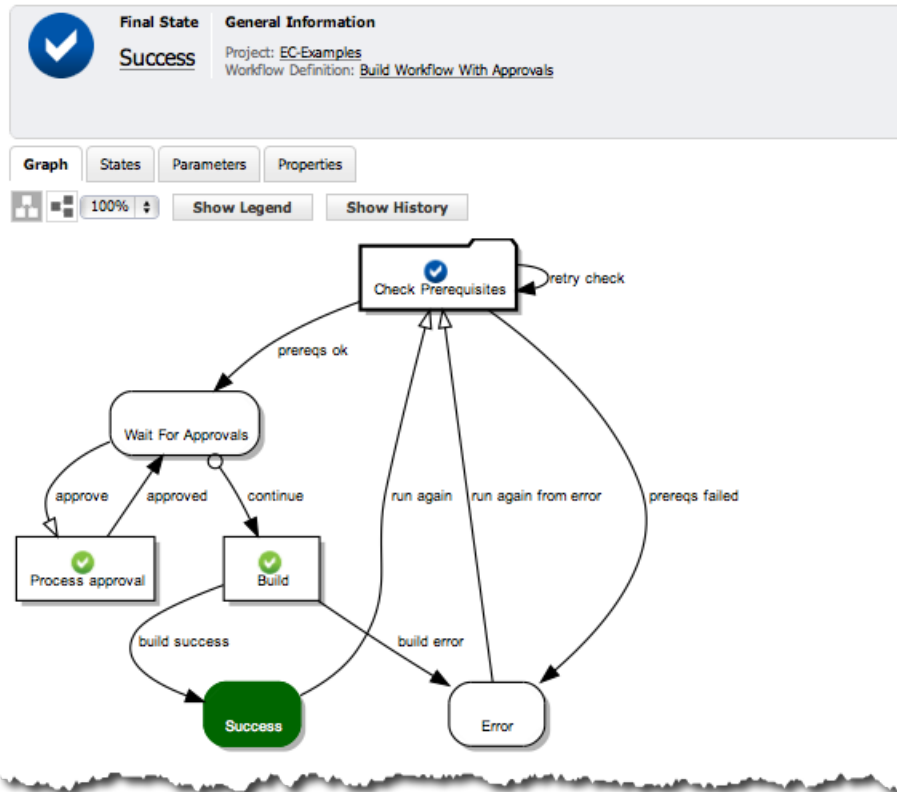
Note: Links to a directory automatically work in Internet Explorer, but if using Firefox, local links are disabled unless the default security policy is modified or an extension is used. Refer to http://kb.mozillazine.org/Links_to_local_pages_don%27t_work for more details.

When the workflow is complete...

Your Workflow Details page will be similar to the following example:

Workflow Details – lassen-33

Run Again



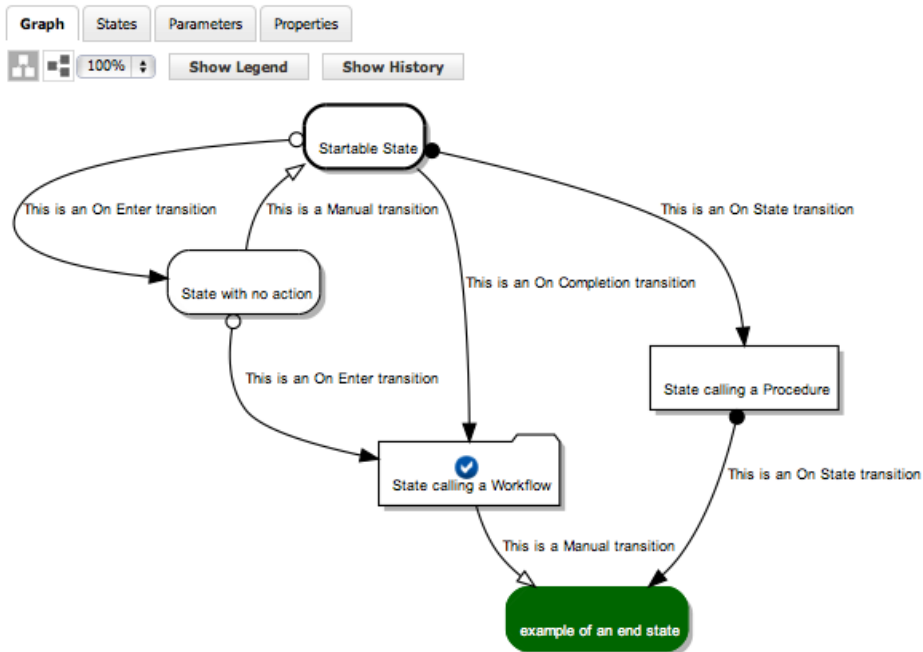
- When the workflow is complete, fewer links are available at the top of the table and the Manual Transitions section is no longer available.
- The "checkmark" icon indicates the workflow is complete. Click the state shown under Final State ("Success" in our example), to open that state's State Details panel.
- The **General Information** section is the same as when the workflow was running.

Graph tab

The Graph tab displays your workflow in a visual format. This format is the default view. The following screen is an example of a workflow graph. Depending on your workflow, your graph could be very simple or much more complex than the example below.

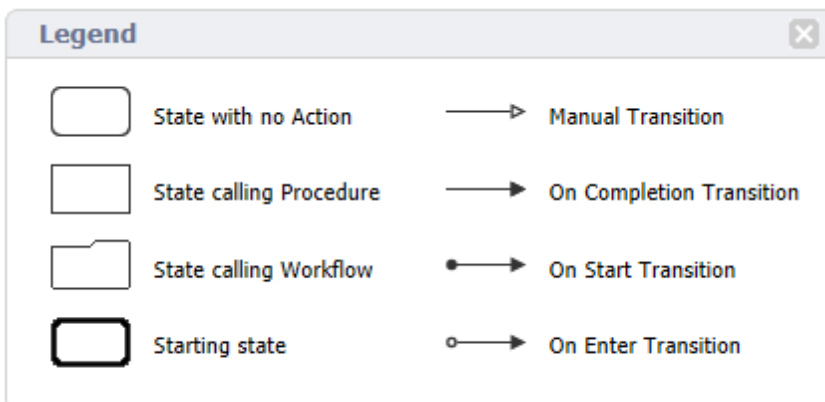
Note: This graph updates in real-time, which means you will see the "active" state highlighted with a color. An icon on the state represents an action.

Use the icons below the Graph tab to rotate the graph for better use of your screen viewing area, and you can re-size the graph using the "%" drop-down menu to choose another size.



Show Legend tab

The graph above illustrates all state and transition types. The state and transition names in this graph are labels for the type of state or transition represented. Use the Show Legend tab to see a quick reminder of state shape and transition line-ending definitions.



Using the graph's "quick-access" features

The following screen example is a portion of a workflow graph.

In the graph, states and transitions are links

- Clicking on a state takes you to the State Details panel to that state's transitions, parameters, and properties.
- Clicking on a transition takes you to the Transition Details panel to see details specifically about that transition, including its condition information.

(Hover your mouse over the transition or transition name and click when the color changes.)

However, if you **right-click** on a state or transition, other link options are available. These options are different for a running or completed workflow versus the options on the Workflow Definition Details page that were available to you while you were defining your workflow.

Available pop-up context-sensitive links include:

- For states, depending on which actions were defined for the state, you may have one or more of the following options:
 - Go to Action - opens the Workflow Details or Job Details page for that state.
 - Go to Action Definition - for states calling a procedure, the Procedure Details page is opened. For states calling a workflow, the Workflow Definitions Details page for the "subworkflow" is opened.
 - Go to State Definition - opens the State Definition panel for that state.
- For transitions, the only link available is: Go to Transition Definition, which opens the Transition Definition panel.

Show History

Selecting Show History allows you to see at-a-glance how your workflow progressed through its states and transitions, or for example, how many times the same state was visited and depending on outcome, which transitions were taken and then the outcome of those target states.


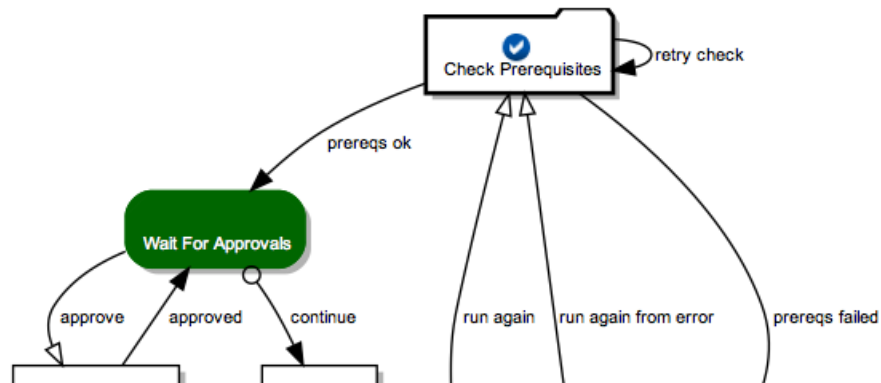
The following example shows a workflow started with the "Check Prerequisites" state and is now at the "Wait For Approvals" state.

Project: EC-Examples

Workflow Details – lassen-139

[Run Again](#) [Stop Workfl](#)

Active State	Manual Transitions	General Information
 Wait For Approvals	approve	Project: EC-Examples Workflow Definition: Build Workflow Wit

[Graph](#) [States](#) [Parameters](#) [Properties](#)
 100% [Show Legend](#) [Show History](#)



So how did the workflow progress through these states?


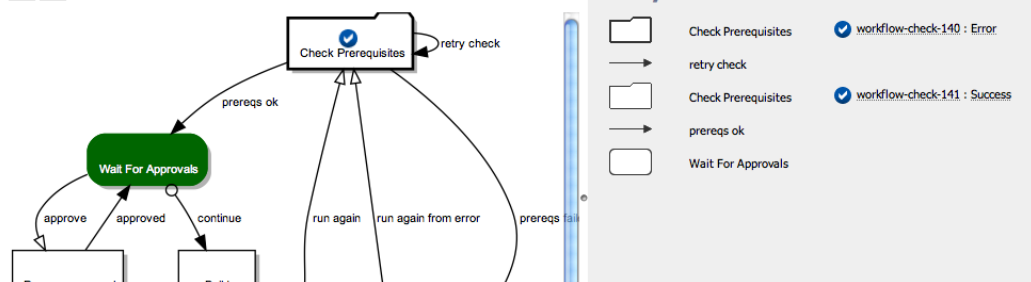
In the next screen example, with the History panel open, you can clearly see the workflow started and the check failed the first time, so the "retry check" transition was taken. The subworkflow was successful the second time, and the workflow has now transitioned to the "Wait For Approvals" state.

Project: EC-Examples

Workflow Details – lassen-139

[Run Again](#) [Stop Workflow](#) [Delete](#) [View Log](#) [Access Control](#) [Stop Refresh](#) [Star](#)






Active State	Manual Transitions	General Information
 Wait For Approvals	approve	Project: EC-Examples Workflow Definition: Build Workflow With Approvals


[Graph](#) [States](#) [Parameters](#) [Properties](#)
 100% [Show Legend](#) [Hide History](#)


Notice the links displayed in the History panel. You can select links for completed jobs or workflows, which display the Job Details or Workflow Details page, respectively.

States tab

Selecting the States tab provides a table containing all state names and actions for this workflow.

Workflow Details – workflow_2044_201204291458  Run Again  Delete  View Log  Access Control 

**Final State**
Approval needed



General Information
Project: Sample
Workflow Definition: Lifecycle

Graph

States

Parameters

Properties

State Name	Action
Build	 BuildProduct-build-85304 [Show All]
Test	 workflow_2045_201204291458 : System tests [Show All]
Failed	
Approval needed	

The following links are available in the States table

- State Name - Click any state name in this column to go to that state's State Details panel. (See the next section for more information about the State Details panel).
- Action - Actions are available in this column only if you are using a subjob or a subworkflow. If so, links in this column connect you to the Job Details or Workflow Details page for the subjob or subworkflow, respectively.
- [Show All] link - If the state executed multiple times during a workflow run, only the last subjob or subworkflow is displayed. However, clicking on Show All displays the results for all subjobs or subworkflows executed during a workflow run.

State Details panel

This panel displays state components, including transitions, parameters, and properties.

- Use the **Access Control** link to set privileges for this state. *For more information*, see the [Access Control](#) help topic.
- The state name, "Build" (in our example), is shown at the top of the panel.
- Job or Workflow status links to the job or workflow created by the action.
- The Transitions tab is open to display the transition table listing all transitions for this state.

Column descriptions

Column Name	Description / Actions
Name	Name(s) of transitions currently defined for this state. Click on a transition name to open the Transition Details panel.
Trigger	The type of trigger currently defined for this transition.
Target	The target state for this transition. Click on a target state name to open the State Details panel for that state.
Condition	This condition was specified when the transition was created. A condition specifies under which circumstances the transition is taken. You can edit this condition on the Transition Definition pane, but if you edit the condition during a running workflow, the change will not take effect until you run the workflow again.
Actions	Access Control - Use this link to set permissions on this transition.

State Details

Build

[Access Control]

No Description

BuildProduct-build-85304

[Show All]

Transitions

Parameters

Properties

Name	Trigger	Target	Condition	Actions
Build failed	onCompletion	Failed		Access Control
Build succeeded	onCompletion	Test	<code>\$/javascript mySubjob.outcome == "success";]</code>	Access Control
retry	onCompletion	Build	<code>\$/javascript mySubjob.outcome == "error" && myState.getProperty("/increment numberOfRetries") < 3;]</code>	Access Control

The **Parameters** tab provides a table listing all defined parameters for this state. You will see any

parameters passed to this state the last time this state was entered, but only if parameters were created on the State Definitions Details panel. This table does not allow you to edit the parameter's value.

These parameters are copied automatically to the top-level property sheet and you will see them in the properties table also.

State Details

Build
No Description

✓ BuildProduct-build-85304 [Show All]

Transitions Parameters Properties

Name	Value
branch	build

Column descriptions

Name - The name of the parameter defined for this state.

Value - The value currently defined for this parameter.

The **Properties tab** provides a table containing all defined properties for this state. If no properties were defined for this state definition, this table will be blank. All defined parameters are automatically copied to the top-level property sheet. Any custom properties defined for the state will be included in this table also.

State Details [X]

Build [Access Control]

No Description

✓ BuildProduct-build-85304 [Show All]

Transitions Parameters Properties

Create Property | Create Nested Sheet | Access Control

Property Name	Value	Description	Actions
branch	build		Delete

Links at the top of the table

- **Create Property** - Click this link to go to the New Property pop-up box to create a new property for this state.

- **Create Nested Sheet** - Click this link to go the New Nested Property Sheet pop-up box to create a nested property sheet.
- **Access Control** - Click this link to add or decrease access control on the property sheet.

Column descriptions

Column Name	Description / Actions
Property Name	Select a property name to edit that property. You can change its name, value, or add/change its description. If the property name is preceded by a folder icon, this is a property sheet. Click on the property name to open the "folder".
Value	The value currently assigned to the property.
Description	A plain text or HTML description for this object.
Actions	Delete - Click this link to delete the property on that row.

Parameters tab

This tab provides a table listing all defined parameters for this workflow's starting state. You will see any parameters passed to the starting state when this workflow was launched, but only if parameters were created on the State Definitions Details page. This table does not allow you to edit the parameter's value.

These parameters are copied automatically to the top-level property sheet and you will see them in the properties table also.

Name	Value
test1	1
test2	2

Column descriptions

Name - The name of the parameter defined for this starting state.

Value - The value currently defined for this parameter.

Properties tab

Select the Properties tab to see the Properties table. If no properties were defined for this workflow definition, this table will be blank.

Create Property Create Nested Sheet Access Control			
Property Name	Value	Description	Actions
property1	full build		Delete
property2	test only		Delete
test1	1		Delete
test2	2		Delete

The links at the top of the table

- **Create Property** - Click this link to go to the New Property pop-up box to create a new property for this state.
- **Create Nested Sheet** - Click this link to go to the New Nested Property Sheet pop-up box to create a nested property sheet.
- **Access Control** - Click this link to add or decrease access control on the property sheet.

Column descriptions

Column Name	Description / Actions
Property Name	Select a property name to edit that property. You can change its name, value, or add/change its description. If the property name is preceded by a folder icon, this is a property sheet. Click on the property name to open the "folder".
Value	The value currently assigned to the property.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
Actions	Delete - Click this link to delete the property on that row.

Workflow Log

This page displays the workflow log.

Notice the name after the page title, "template-1257" — this is the name of the workflow, which includes the workflow name plus the Commander auto-generated ID number, and it is the object of the workflow log in our example.

Links and actions above the table

- Drop-down menu - Use the down-arrow to select Error, Warn, or Info. The first time you choose a severity level, it will become your default level—the one you always see first when you view this page. Selecting another level changes your default view.
- The "star" icon allows you to save this workflow definition to your Home page.
- The "bread crumbs" *Project: SoftwareBuild / Workflow: template-1257* provide links to a previous web page.

Project: SoftwareBuild / Workflow: template-1257

Workflow Log – template-1257

INFO

7 Results

Time	Severity	User	Subject	Message
2010-12-06 15:12:42 PST	INFO	admin	wip	User 'admin' has completed the workflow
2010-12-06 15:12:28 PST	ERROR	project: SoftwareBuild	wip	Failed to start the sub-job for state 'wip': Missing parameter(s) to 'wf_start_wip': plugin, version
2010-12-06 15:12:28 PST	INFO	project: SoftwareBuild	wip	Sending onEnter email notifiers for state 'wip'
2010-12-06 15:12:28 PST	INFO	admin	start_work	User 'admin' is taking manual transition 'start_work' from state 'stable' to state 'wip'.
2010-12-06 14:55:30 PST	INFO	project: SoftwareBuild	stable	Sending onEnter email notifiers for state 'stable'
2010-12-06 14:55:30 PST	INFO	project: SoftwareBuild	stable	Starting state: 'stable'
2010-12-06 14:55:30 PST	INFO	admin	workflow_53	Created workflow 'workflow_53'

Records per page: 20 1 thru 7 of 7

Column descriptions

Time - Displays the time when an event occurred that caused the server to generate a message.

Severity - the three severity levels are:

- **ERROR** - An unexpected failure was encountered while entering a state or launching a sub-action. Generally, this error indicates a critical problem with the workflow that requires fixing the workflow definition.
- **WARN** - A non-critical issue was encountered while the workflow was running.
- **INFO** - Provides workflow activity information including the state entered, transitions taken, and so on.

User - 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.

Subject - Objects displayed in this column are the subject of the message. These objects are linked to either the Workflow Details or the State Details page.

Message - A text message generated by the Commander server while the workflow was running.

Transition Workflow


Use this page to transition your workflow to a different startable state.

Note the name after the page title, "template-99"—this is the name of the workflow in our example.

Links and actions above the table

- The "star" icon allows you to save the Transition Workflow page to your Home page.
- The "bread crumbs" *Project: SoftwareBuild / Workflow: template-99* provide links to a previous web pages.

Project: SoftwareBuild / Workflow: template-99

Transition Workflow – template-99 

Transition: start_work

State: stable

Target State: wip

Parameters version: Required

Field descriptions

Fill-in the fields as follows:

Field Name	Description
Transition	The name of this transition.
State	The name of the state that owns the transition— the active state of the workflow.
Target State	The name of the target state— the state where the workflow will transition.
Parameters	In the screen example above, "version" is the name of the parameter and you must supply a value for this parameter. This is a required field.

Click **OK** to transition the workflow to the target state.

Workspaces

This page displays all workspaces currently available to ElectricCommander. At a glance, you can see all paths to every workspace.

- To create a new workspace, click the **Create Workspace** link.
- To find a workspace, use the **New Search** link.

Column descriptions

Column Name	Description / Actions
Workspace Name	The the name created for this workspace. Click on a Workspace Name in this column to edit that workspace.
Zone	The name of the zone where this workspace resides.
Enabled	This workspace is enabled if this box is "checked". Note: When this box is checked, the workspace is enabled, which means it can be accessed. In the case where a job or job step cannot access the workspace, the job will "queue", waiting for the workspace to become available.
Local	If the box is "checked", the workspace is "local", which means files in the local workspace are accessible only from the resource that originally created the file.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> .
Drive Path / UNC Path / UNIX Path	The path to the workspace.
Actions	Copy - Use this link to make an exact duplicate of an existing workspace, then select the copy to go to the Edit Workspace page to change the workspace name. Delete - Use this link to delete the workspace on the same row.

Note: Add access control privileges to a workspace on the Edit Workspace page.

Workspace - create new or modify existing workspace

To create a new workspace

Fill-in the fields as follows:

Field Name	Description
Name	Supply a unique name to distinguish this workspace from other workspace names.
Description	A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
Enter three paths to use on resources to access the workspace root directory:	
UNC Path	A path in UNC notation (such as <code>//server/ec/a/b/c</code>) to use on Windows machines. The path can use either slashes or backslashes for separators.
Drive Path	A second path for Windows machines, based on drive notation such as <code>N:</code> or <code>N:/b/c</code> . If the drive path contains more than just a drive letter, additional directory names must match the trailing directories of the UNC path. For example, if the UNC path is <code>//server/ec/a/b/c</code> , then <code>N:/b/c</code> can be used as the drive path (agents automatically map <code>N:</code> to <code>//server/ec/a</code>), but <code>N:/x/c</code> is not allowed.
UNIX Path	The path to use for UNIX machines, typically based on an NFS mount. You must ensure appropriate mounts were created on all machines where the path will be used. <i>Commander does NOT automatically mount filesystems for UNIX.</i>
<p>Note: If the workspace will be used only on UNIX machines, you can omit the UNC and drive paths. Conversely, if the workspace will be used only on Windows machines, you can omit the UNIX path.</p> <p>For more information, see the Workspaces and Disk Space Management Help topic.</p>	
Zone	Use the drop-down arrow to select a zone for this workspace.
Local	Select this checkbox if the workspace is "local", which means files in the local workspace are accessible only from the resource that originally created the file.

Field Name	Description
Enabled	"Check" this box to enable this workspace. When this box is checked, the workspace is enabled, which means it can be accessed. In the case where a job or job step cannot access the workspace, the job will "queue", waiting for the workspace to become available.
Credential Project and Credential Name	If you need to supply a user name and password to mount the workspace, you must supply the credential project name and then choose the appropriate credential name, which contains the required user name and password. Click the Browse link to select information for these fields.

Click **OK** to save your workspace specification and see it listed on the Workspaces page.

If you need to add properties or privileges to this workspace:

Click **OK** to create the workspace and return to the Workspaces page, then click on the workspace name to go to the Edit Workspace page.

To edit an existing workspace

All previously supplied workspace information is available on this page and you can modify existing information or add new properties or privileges.

- Click the **Access Control** link [at the top of the page] to add access privileges to this workspace.
- To assign a new workspace name, highlight the existing name and type-in a new name.
- Re-assign paths if necessary.
- Use the drop-down menu to select a different zone if necessary.
- Add or modify the Credential Project name or the Credential Name if necessary.

Click **OK** to save your modified information.

If Custom Workspace Properties were already assigned, you may modify these properties or add new ones. If a property name is preceded by a "folder" icon, this denotes a Nested Sheet.

Click the "star" icon at the top of the page to add this page to your Shortcut section on the Home page.

For more workspace information, see the [Workspace and Disk Space Management](#) Help topic.

Workspace file

This page displays a file from the workspace for a job, typically the log file for a job step. For log files, this file contains both standard output and standard error from the step.

If you view a file while the job step generating the file is still running, the page refreshes automatically as information is added to the file.

Tip

The log does not contain output you were expecting?

Check the job step commands. If the step redirects its output to a different location, information will not appear in the log file.

Tutorials

The ElectricCommander help topics in this directory help you learn more about ElectricCommander and specific ElectricCommander tasks.

You can directly access these help topics from the help system Table of Contents.

You can also access these help topics while using ElectricCommander. Click **Help** at the top-right corner on any ElectricCommander web page.

Adding a link to a job

Commander provides an easy method for adding links to the Job Details page to view files and reports for any job where you need to access additional information quickly. Adding direct links eliminates logging into where ever these files might be stored.

Note: The Workflow Details page and several other Commander web pages allow adding links also.

This tutorial shows how to add links to a job and have them immediately available on the Job Details page.

An explanation of what you see:

- The Procedure Details page for the Adding a link to a job procedure contains 2 steps: Create File and Create Job Link
- The Action column displays a code fragment of the script used to create the `readme.txt` file and a reference for the Create Job Link step. For this example, we want to link to a "readme" file.
(You can click on the Create File step name to go to the Edit Step page to see the full ec-perl script shown partially in the Action column.)

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page, where the new link to the `readme.txt` file is displayed under the Links heading in the summary section at the top of the page.



Implementation

Use the following instructions if you would like to practice creating a link on the Job Details page now.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander. These instructions only coach you through the process you would use to create links on your Job Details pages.

1. Using the same Adding a link to a job procedure, click the **Create Step** link.
2. On the Choose Step Type dialog, select the Subprocedure tab.
 - Select Project.
 - Place your cursor in the text field and start typing EC-Utilities.
 - For Procedure, click inside the text box and select the Create Job Link procedure.
 - Click **Create**.
3. On the New Step page, supply `Google search` for the new step Name.

4. In the Parameters section:
 - Link Label - supply `Google Search`.
(This is any name you choose for the name of your link.)
 - Link Location - supply `http://www.google.com`
(This is the URL to your link location or it can be a relative path to where you want to create a link.)
 - Click **OK** to save your entries and return to the Procedure Details page.
5. On the Procedure Details page, click **Run**.
6. On the Job Details page, as soon as the links are processed you will see the new "Google Search" link in addition to the "readme.txt" link.

Clicking on the new Google Search link will take you to the Google website.

Related information

[Job Details](#) - see this help topic for more detailed information about creating links and complete information on other aspects for using the Job Details page.

Calling a subprocedure

The ability to create *reusable components* is a major ElectricCommander strength. Procedures in any project or plugin can be called as a subprocedure in a step for another procedure. Creating a subprocedure is one way of reusing existing functionality already implemented in Commander.

This tutorial shows how to use the step configuration process to call an existing procedure to use as a subprocedure.

An explanation of what you see:

- The Procedure Details page for the Calling a subprocedure procedure contains a step named Execute a procedure from this project.
(Clicking on this step takes you to the Edit Step page.)
- The Action column displays a plugin (or project) name, EC-Tutorials, and a procedure name, Working with properties in a stored procedure.
(Clicking on the plugin/project name takes you to the Project Details page for that project, and clicking on the procedure name takes you to the Procedure Details page for that procedure.)

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Use the following instructions if you would like to practice creating another subprocedure for this "Calling a subprocedure" procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander and cannot be transferred to your procedures. These instructions only provide a clearer explanation for how you can call subprocedures in your projects.

To create a step that executes a subprocedure:

1. On the Procedure Details page, click the **Create Step** link to see the Choose Step Type dialog.
 - Select the Subprocedure tab.
 - Select Plugin and place your cursor in the text field and start typing EC-Tutorials.
 - For Procedure, click inside the text box and select Reserving a Resource for job step duration procedure or whichever procedure you would like to choose.
 - Click **Create** to go to the New Step page to create the step to call an existing procedure to use as a subprocedure.
2. On the New Step page, supply any new step name of your choice.
Notice that no parameters are required for this step.
3. Click **OK** to return to the Procedure Details page to see your new step in the table, then click **Run**.
4. When the Job Details page is displayed, you can see your new step and the procedures it is calling.

Related information

[Procedure - create new or edit existing procedure](#) - help topic

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Checking outcome of preceding step

Commander steps can have a *success*, *warning*, *error*, or *skipped* outcome. Sometimes it is useful to execute or skip subsequent steps based on the outcome of a previous step. Used with error handling behavior available at the step level, sophisticated flow control can be achieved easily.

This tutorial demonstrates how to see the outcome of a preceding step and use run conditions to execute steps based on that outcome.

An explanation of what you see:

The Procedure Details page for the procedure, "Checking outcome of preceding step", contains 3 steps:

- `step 1 - force an error` - This step forces an error by attempting to run on a resource that does not exist.
This action was achieved by hard-coding the step named `"step 1 - force an error"` to run on a resource called `"this-does-not-exist"`.
- `step 2 - execute if step 1 failed` - This step uses JavaScript to check the outcome of the first step and executes if the first step failed.
You can view the JavaScript for the run condition by clicking on the step name to take you to the Edit Step page—see the Run Condition field in the Advanced section.
This statement checks the result of `"step 1"` for an error. If there is an error, `"step 2"` will execute.
- `step 3 - execute if step 1 succeeded` - This step also uses embedded JavaScript to check the outcome of the first step and executes if the first step succeeded.
This is a slight variation of `"step 2"`. Instead of looking at `"step 1"` for an error, this step checks for success and executes only if `"step 1"` succeeds.
You can view the JavaScript for the run condition by clicking on the step name to take you to the Edit Step page—see the Run Condition field in the Advanced section.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Use the following instructions if you would like to practice creating steps to check the outcome of a preceding step.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander and you cannot transfer any steps from this tutorial to your projects. These instruction are provided only to give you a clearer definition for how to recreate this process in your own projects.

- Click the **Create Step** link.
 - In the Choose Step Type dialog, choose Command step to go to the New Step page.
 - Supply a Name for your step.
 - In the Command(s) text box, you can use the information in the above section. "Copy and paste" the provided JavaScript, changing the step name (between the brackets) to the new step name you supplied.
- Repeat this process to create 3 steps, clicking **OK** to save each step.

- When your 3 new steps are displayed on the Procedure Details page, click **Run** to see your steps running on the Job Details page.

Related information

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page.

Click the **Help** link in the upper-right corner of any Commander page to access page help.

Conditional execution

Process automation frequently contains actions to be executed only if certain conditions are met. Conditional execution can be applied directly to your procedures rather than having to maintain larger scripts to handle these actions. For example, you may need to control whether or not to build on different operating system types.

This tutorial shows you how to implement simple conditional execution in a step.

An explanation of what you see:

This procedure defines one parameter and then uses a run condition to check the parameter value to decide whether or not to execute a step.

- The Procedure Details page for the Conditional execution procedure contains a step named step 1. (Clicking on this step takes you to the Edit Step page.)
- The Action column displays the command to be executed.
- The Parameters section displays a parameter with values.
The parameter named "Execute step 1" was created as a checkbox.

If checked, the parameter equates to "true".

In the run condition for "step 1", the parameter is read using the following property reference notation:

```
${Execute step 1}.
```

(Clicking on the Parameter Name takes you to the Edit Parameter page to make modifications you may need.)

Click **Run** (at the top of the page) to run this sample procedure, which then takes you to the Run Procedure page where you could cause `step 1` to execute by selecting the Execute step 1 checkbox.

- Click **Run** again to go to the Job Details page to see the job created by running this procedure.

Implementation

Review the following instructions if you would like to see how to reproduce a conditional execution step.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander. These step-by-step instructions are provided only for more detailed information so you can easily see how to create your own "conditional execution" solutions within your procedures.

- Click the **Create Parameter** link to go to the New Parameter page.
 - Supply a new parameter name.
 - Click the Type down-arrow and select Checkbox.
 - Select Initially checked?
 - For Default Value, type "true".
 - Select Required?
 - Click **OK** to save your parameter and return to the Procedure Details page.
- Click the step name to go to the Edit Step page.

- Change the Run Condition text to check the new parameter name you supplied above.
- Click **OK** to return to the Procedure Details page.
- Click **Run** to go to the Run Procedure page to see 2 parameters where you can now decide if both are required or not.
- Click **Run** to go to the Job Details page.
 - If only 1 parameter was "checked" as required, you will see the "skipped" notation in the Status column.

Related information

More complex run conditions can be implemented using inline JavaScript.

See the tutorial, "Checking outcome of preceding step" for a more complex run condition example.

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Custom parameter layouts

By default, parameters for procedures are displayed in alphabetical order based on the parameter name. Using the `ec_parameterForm` property allows parameters to be displayed in a specified order, different from the default, alphabetized order by name.

This tutorial shows how to create a custom parameter layout for procedures.

An explanation of what you see:

- This procedure implements 2 parameters.
- The Action column contains the command to execute. In this case, the action shows the `ec_parameterForm` property which was attached to the procedure.
- Notice the checkmarks in the Req? column, which indicates both parameters are "required" each time this procedure runs.
- In the Custom Procedure Properties table, notice the `formElement` values supplied to determine the new parameter order.
 - If the procedure is run **without** the `ec_parameterForm` property, the parameter input form displays parameters in default, alphabetical order:

Parameters

first parameter:

second parameter:

Click **Run**.

On the Run Procedure page, notice you are presented with a Parameter section that shows the 2 parameters in reverse order because values supplied for the `ec_parameterForm` property were called to create this new order.

Parameters

this is the label for "second parameter"

this is the label for first parameter

The following XML creates the new order for these 2 parameters:

```
<editor>
  <formElement>
    <property>second parameter</property>
    <label>this is the label for "second parameter"</label>
```

```

    <type>entry</type>
    <required>1</required>
  </formElement>
  <formElement>
    <property>first parameter</property>
    <label>this is the label for first parameter</label>
    <type>entry</type>
    <required>1</required>
  </formElement>
</editor>

```

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Note: This tutorial is intended for viewing, but not modifications within its procedure, and any changes you might make will not be saved when you upgrade Commander.

However, in your own projects, you can use the sample XML above to change the parameter order in any of your procedures.

If you copy and paste the code samples we provided, using the `ec_parameterForm` property in one or more of your procedures, make sure you replace our example values with your parameter names, labels, and so on. If you have more than 2 parameters to reorder, create any number of additional `<formElement>` sections you need.

Related information

[Customizing the Commander UI /Customizing Parameters](#) - help topic

[Step - create new or edit existing step](#) - help topic

[Parameter - create new or edit existing parameter](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Email notifications

Use email notifiers to send information to people who need or want ElectricCommander notifications, whether or not they actually use Commander. For example, you might want to set up an email notifier to send log file error excerpts matched by *postp* to a team or individual responsible for investigating the error.

This tutorial illustrates how email notifiers can be attached to steps to automatically send an email when jobs, steps, or workflows execute.

Prerequisite: If you do not have an Email Configuration, you must create one before you can set up an email notification. An Email Configuration tells the Commander server which host to use for mail delivery. Remember the email configuration name you choose because you will need it later.

An explanation of what you see:

- A step named "do something" that puts a short message in the log file
- A parameter that defines the email address for where you want to send the notification. This parameter is used as the destination value for the notifier named "start notifier". This notifier sends an email when the procedure runs.
- An email notifier named "start notifier" with its Type specified as "onStart".
 - Click on the notifier name to view the notifier details.
 - Make sure the value in the Email Configuration field matches the email configuration name from the "Prerequisite" section above.
 - Click **OK** to save any changes and return to the Procedure Details page.

Click **Run**.

On the Run Procedure page, in the Parameters section, supply an email address, an address list, or email group name.

Click **Run** again to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Note: Use the following information to create an email notifier in your own project, assuming you have already setup an email configuration. Any changes you might make within this tutorial will not be saved when you upgrade Commander.

Using the Commander UI to add an email notifier:

1. On the Procedure Details page, click on the Create On Start Email Notifier link.

Procedure Details – Email Notifier E:

Procedure Steps

Currently, there are no records to display in this list.

Parameters

Currently, there are no records to display in this list.

Email Notifiers

Currently, there are no records to display in this list.

Custom Procedure Properties

Currently, there are no records to display in this list.

2. Fill-in the fields on the New On Start Email Notifier page. See the example below.

New On Start Email Notifier

Name:

Description:

Condition:

 Leave empty to always send an email or use a run condition to only send an email when certain conditions are met

Formatting Template:

 Subject: Commander notification

 Message body goes here.

Email Configuration:

Destinations:

Note: If the Email Configuration field is left blank, the system uses the email configuration with the name "default".

Click **OK** to save the notifier.

When the procedure is run, an email notification is sent to the email address(s) specified in the Destinations field. In this tutorial example, the Destinations field refers to the parameter value passed in to the procedure when it is run.

The value is referenced using the following notation:

```
$/myJob/Send notification email to]
```

Related information

[Email Configuration - create new or edit existing email configuration](#) - help topic

[Email Notifier - create new or edit existing email notifier](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Executing tasks on all resources in a pool

Creating resource pools (grouping resources) means you can reference many resources using one name, the pool name. When configuring a step, you can "broadcast" the step to all resources in a pool. For example, you might want to start an application deployment at the same time on all resources in a pool.

This tutorial shows how to group resources into a resource "pool".

An explanation of what you see:

This procedure implements 3 steps:

1. The first step, `Setup environment`, creates a pool for resources.

Clicking on the step name takes you to the Edit Step page to see the full command text supplied to create this step—a partial section is displayed in the Action column.

Note: Functionality to create this step is beyond the scope of this tutorial. For more information, see the link to "Using ectool and the Commander API" below.

The following resources were created and added to a pool named EC-Tutorial-pool:

EC-Tutorials-resource1

EC-Tutorials-resource2

EC-Tutorials-resource3

2. The second step, `Execute command on all resources in pool`, executes an echo statement on the resource EC-Tutorial-pool and has the "broadcast" option selected, which results in the step executing on all resources in the pool simultaneously.
3. The third step, `Cleanup`, deletes the resources created in the `Setup environment` step.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Note: This tutorial is intended for viewing only and/or clicking on the available links to see more information. Any modifications you might make within this tutorial cannot be transferred to your projects and will not be saved when Commander is upgraded.

Before you can create the steps illustrated in this tutorial procedure for your own purpose, you must have already defined your resources and created resource pools.

See the following list of related links for more help with creating and using resource pools.

Related information

[Using ectool and the Commander API](#) - help topic

[API commands](#) - help topic

[Resources](#) - help topic

[Resource Pool - create new or edit existing pool](#) - help topic

[Resource Pool Details](#) - help topic

[Step - create new or edit existing step](#) - help topic

Factory procedures

Factory procedures allow dynamic behavior in process automation. To implement a factory procedure, you must first create a procedure that other procedures can use as a template. A procedure uses the template to construct new procedures at run time. Instead of creating steps or complex scripts to account for all possible scenarios, procedures can be created "on the fly" based on user input, resulting in a very dynamic system.

This tutorial shows how to create factory (dynamic) procedures, which are procedures that modify themselves at runtime based on parameters passed into them.

An explanation for what you see and what is actually happening:

This tutorial contains 2 procedures and creates a third procedure "on the fly":

- "Factory procedures" - this procedure uses the procedure template (Utility Procedure - Factory Template).
This procedure has a parameter called "number of steps". This parameter takes an integer and is used to specify how many new steps should be generated.
Note: For this tutorial example, only one procedure is constructed to use the template procedure, but you may have multiple procedures that use the factory template procedure.
- "Utility Procedure - Factory Template" - you do not see this procedure on this Procedure Details page, but it is included in the EC-Tutorials project. This is the factory procedure template used when generating new steps in the main procedure. This procedure implements a single step that writes some text to the log file.
This procedure is designed to do a repetitive task, repeating the task a variable number of times. For example, you might need to run a test framework "n" number of times.
Note: You can view this procedure in the EC-Tutorials project.

The main procedure, "Factory procedures", implements 3 steps:

- The first step: "create dynamic procedure" is a Perl script that uses the Perl API to create a new procedure.
The following script creates a new procedure containing the number of steps specified by the parameter:

```
my $procedureName = "Dynamically Created Procedure ${jobId}";
$ec->createProcedure("${myProject/projectName}", $procedureName);

for(my $i=0;$i<${number of steps};$i++)
{
    $ec->createStep("${myProject/projectName}", $procedureName, "Generated step
    $i", {subprocedure => "Utility Procedure - Factory Template",
        actualParameter => {actualParameterName => 'text', value => "Executing
        generated step $i"}});
}

$ec->setProperty("/myJob/dynamicProcedure", $procedureName);
```


1. First, the procedure generates a unique name for the new procedure to be created. The name is created by appending the value of the `${jobId}` property string.

2. A new procedure is created in the current project.
 3. Based on the number of steps to be created that were passed in as a parameter, a number of subprocedure steps are created in the procedure. The subprocedure steps call the utility procedure associated with this tutorial.
 4. The name of the newly created procedure is stored in `/myJob/dynamicProcedure`
- The second step: "Run dynamically created procedure" calls the procedure that was created in Step 1. Step 2 appears to call nothing, but in fact it is calling a procedure referred by `$/myJob/dynamicProcedure`.

Click **Run** to run this sample procedure and go to the Job Details page.

Running the procedure and specifying the number of steps to be generated as 3, results in the following job being executed:




Job Details – Recipe 15 - Factory Procedures-11

**Completed with Success**
Start Time: 2011-12-15 14:44:32 PST
Elapsed Time: 00:00:03.620

General Info
Project: EC-Tu
Procedure: Re
Launched by:
Priority: norm.

Steps | Diagnostics | Parameters | Properties | Notifiers | Published

View: All ▾

Step Name	Log	Status
create dynamic procedure		✓ Success
<input checked="" type="checkbox"/> Run dynamically created procedure		✓ Success
<input checked="" type="checkbox"/> Generated step 0		✓ Success
display text		✓ Success
<input checked="" type="checkbox"/> Generated step 1		✓ Success
display text		✓ Success
<input checked="" type="checkbox"/> Generated step 2		✓ Success
display text		✓ Success
Delete dynamically created procedure		✓ Success

Notice the steps that execute in parallel in the dynamically created procedure:
Generated step 0, Generated step 1, and Generated step 2

Implementation

To create a "factory procedure" process in your project:

1. Create a utility procedure - this procedure performs a task.
 - Create the steps you need for your task.
2. Create a second procedure - the "factory procedure," providing a name of your choice for this procedure.
 - Create a minimum of 3 steps in this procedure.
 - step 1 - create a new procedure that contains steps, each calling the utility template procedure.
 - step 2 - this step executes the procedure created in step 1.
 - step 3 - this step deletes the procedure created in step 1.

Each time you run the "factory procedure," it creates a temporary procedure to run the utility procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page.

Click the **Help** link in the upper-right corner of any Commander page to access page help.

Postp extension

postp is a powerful Commander feature (postprocessor) you can use to monitor step output in real-time and take action. Frequently, the *action* is to extract information from step output, store it in the Commander database, and set properties for reporting. In addition, *postp* can be used to execute functionality depending on whether or not a pattern is found. *Postp*, an essential part of the Commander toolbox, provides fast feedback to users about job step status.

Note: *Postp* matchers are written in Perl. *Postp* extensions are either loaded from a file located on the Commander server or from a stored property.

This tutorial demonstrates how to construct a simple custom *postp* matcher to extend *postp* capabilities, and loads the *postp* extension from a property.

An explanation of what you see:

- This procedure implements a single step, "get directory listing", that retrieves a directory listing using a simple Perl script. Perl is used for the example to ensure the same output no matter which operating system is used for this tutorial. Understanding the Perl code used to generate the tutorial listing is not required for this tutorial.
- A property was created. This procedure has a *postp* extension attached in a property named `matcher` and contains the following Perl code:

```
use ElectricCommander;
push (@:gMatchers,
{
  id => "fileCountMatcher",
  pattern => q{(\d+ File\s\)},
  action => q{
    setProperty("summary", "Matcher $matcher->{id} found the following
    output\n\n$1");
  }
});
```

The *postp* matcher has an identifier ("id"), a regular expression ("pattern") that matches text and an "action" to be performed when text matching the regular expression is found. In this case, the regular expression "id" is `fileCountMatcher`, and an example of the output being matched is "2 File(s)". When the pattern is matched, the action to perform is to set the summary property to the text matched by the regular expression.



The custom *postp* matcher is loaded using the following command in the Postprocessor field when creating the step:

```
postp -loadProperty /myProcedure/matcher
```

Click on the step name to go to the Edit Step page to see how this step was created—note the text in the Command box and the Postprocessor field.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

See the Status column. The number of files in the file system root where the workspace is located is displayed.

Steps	Diagnostics	Parameters	Properties	Notifiers	Published Artifact Versions	Ret
View: All ▾						
Step Name	Log	Status				
get directory listing		 Matcher fileCountMatcher found the following output 4 File(s)				

Implementation

To try using postp functionality in your own project:

- Edit an existing procedure or create a new one with a "matcher" as a property.
For your practice example, you can "copy and paste" the matcher example provided in this tutorial (above).
- Create a new Command step for this procedure - filling in the Postprocessor field on the New Steps page.
Again, for practice, you can use the postp string provide above in this tutorial.

When you run your new procedure, you will see results similar to those you saw after running the tutorial example procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

[Postprocessors](#) - help topic

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Publishing and retrieving an artifact

To ensure traceability in processes, output needs to be versioned and tracked as it is handed from one process stage to another. With Commander Artifact Management, implementing your processes is auditable and secure.

This tutorial shows how to produce and consume artifacts as part of a Commander process.

An explanation of what you see - the following concepts are illustrated in this procedure:

- Execute a simple `ec-perl` script to create a file to use as an artifact
This is a "Setup" step that creates a text file in a workspace that can be added to an artifact.
Click the Setup step name to go to the Edit Step page to see the full "dummy" script, used for setup, in the Command text box. This script creates a file named `myartifact.txt`.
Note: There is no code specific to Commander in this step, so understanding the Perl code used to generate the text file is not necessary.
- The "Publish artifact" step calls the EC-Artifact:Publish procedure
This step is configured to place the file `myartifact.txt`, created in the Setup step, into an artifact.
To do this, the EC-Artifact: Publish procedure requires 3 other pieces of information:
 - The artifact where you want to publish. This is in the format `Artifact Group:Artifact Key`.
In this case, the name of the artifact (where you want to publish) is `EC-Tutorials:MyArtifact`.
 - The artifact version being published. The version is in the form of `major.minor.patch-qualifier-buildnumber` or `major.minor.patch.buildnumber-qualifier`. For this tutorial, the version is set to `1.0.${jobId}-tutorial`.
Using the `${jobId}` property expansion results in a new artifact being created every time the procedure is run.
 - The repository server from which you need to retrieve the artifact. This tutorial uses a repository named `default` that was created when Commander was installed.
Note: To see how the above information is specified in this step, click the "Publish artifact" step name to go to the Edit Step page.
- The "Retrieve artifact" step calls the EC-Artifact: Retrieve procedure
This procedure takes two required parameters and several optional parameters (see the Edit Step page/Parameter section for this step).
The required parameters are:
 - Artifact - `EC-Tutorials:MyArtifact` (this is the name of the artifact to retrieve)
 - Version - `1.0.${jobId}-tutorial` (this is the artifact version to be retrieved - either an exact version or a version range).
This tutorial specifies an exact version.
- The last (4th) step in this procedure contains the retrieved artifact location.
Note: An important parameter is "Retrieved Property Artifact Location", which is:
`/myJob/retrievedArtifactVersions/${assignedResourceName}` - the location where the artifact files are put after retrieval is stored in this property. By default, the above location is where retrieved artifact versions are stored.
 - Display the location where the artifact is to be retrieved
To do this, retrieve the property value specified in the "Note" above.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

Implementation

To publish and retrieve artifacts, apply these concepts to your artifact projects, creating the steps as required to your procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

[Artifact Management](#) - help topic

[Artifact Details](#) - help topic

[Job Details](#) - help topic

[Publish Artifact Version step](#) - help topic

[Retrieve Artifact Version step](#) - help topic

Reserving a resource for job step duration

Sometimes you may want a resource to perform work for only one job at a time. Because a Commander agent is capable of executing steps for many jobs on a resource, this can produce undesirable performance results. For example, you might want a dedicated resource to stress test an application after you have Commander initiate a load testing task.

This tutorial shows how to reserve a resource for one job step only while that job step is running.

An explanation of what you see:

- This procedure locks the resource for the step and executes the tasks.
- To demonstrate this functionality, two steps are implemented.
 - Notice that both steps are set to run in parallel on the local resource.
 - Both steps request a lock of the resource for the duration of the step execution.

The functionality implemented in each step:

- A simple Perl sleep statement to force a lock over a period of time when the procedure is run
- Select "Step" in the Retain Exclusive field in the Advanced section on the New/Edit Step page

Click a name in the Step Name column (to go to the Edit Step page for that step). You will see the Perl sleep statement added to the Command text box, and "Step" selected on the Retain Exclusive field.

Instead of running the steps in parallel as you may have previously requested, the steps will now execute serially. The second step must wait for the resource to become available because it was reserved by the first step.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

Implementation

Create/Edit Command Steps for your procedures if you need this functionality.

- Go to the Procedure Details page for the procedure where you want a step to reserve a resource.
- Add a Command Step to your procedure (use the step creation link at the top of the table), or edit an existing step.
- In the Command text box, supply a script if you want to specify any time constraints.
- On the New/edit Steps page, in the Advanced section, choose "Step" in the Retain Exclusive field.

The next time you run this procedure, your job step will have exclusive use of the resource you specified.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Running steps in parallel

The main advantage to running steps in parallel is increased performance. You can set a step to run serially as a synchronization point and then execute more steps in parallel. For example, running steps to execute tests on different platforms concurrently would mean numerous tests could complete in the same time period.

This tutorial demonstrates how to run steps in parallel.

An explanation of what you see:

This procedure contains 5 steps:

- Step 1 and 2 run in parallel.
- The third step (sync) is a sync point where the procedure waits until steps 1 and 2 finish executing.
- Steps 4 and 5 execute in parallel after the sync point.
- Steps 1,2,4 and 5 execute simple Perl scripts that print a message and "sleep" for a period of time.

Note: Using "sleep" is not required, but used here to make sure each parallel step runs long enough to be visible while running this tutorial.

Click **Run** to run this sample procedure and see the resulting job status on the Job Details page.

Implementation

Note: Use the following information to create steps to run in parallel in your procedures. Any changes you make within this tutorial will not be saved when you upgrade Commander.

Steps will run in parallel if you select the Run in Parallel option in the Advanced section on the Step - create new or edit existing step page.

Advanced

Run Condition:	<input type="text"/>
Error Handling:	<input type="text" value="Procedure continues, but overall status"/>
Time Limit:	<input type="text"/>
Run in Parallel:	<input checked="" type="checkbox"/>
Always Run Step:	<input type="checkbox"/>
Broadcast:	<input type="checkbox"/>
Retain Exclusive:	<input type="text" value="None"/>
Release Exclusive:	<input type="text" value="None"/>
Shell:	<input type="text" value="ec-perl"/>
Workspace:	<input type="text"/>
Working Directory:	<input type="text"/>
Log File:	<input type="text"/>

Related information

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Step timeouts and steps that always run

Complex processes can be brittle so Commander provides a way to account for different errors in your procedures. You can specify timeouts, error handling methods, and steps that always run no matter what errors have occurred in preceding steps in the procedure. These facilities can make sure your build does not "hang" in a situation where you would prefer the build to continue.

This tutorial shows how to construct a procedure so processes are terminated after a set time period and how to implement a step that always runs in the event of an error (to perform cleanup).

An explanation of what you see:

This procedure implements 2 steps:

(Clicking on the step name takes you to the Edit Step page to see how the step was created.)

- The "force timeout" step has a 10-second timeout and is set to abort the procedure and terminate running steps in the event of a timeout or error.

This step executes a sleep command using `ec-perl`, which makes the step execute for 15 seconds. Step error handling is set to "Abort procedure and terminate running steps". This step will always fail because the 15-second sleep time is longer than the 10-second timeout.

- The "always run" step set to run no matter what the error handling behavior is for any preceding steps.

This step uses an echo statement to output some text. The step property, `Always Run Step`, is set to "true" so this step runs regardless of the result for the preceding step.

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

On the Job Details page: Notice that the "always run" step completed with Success while the "force timeout" step completed with a TIMEOUT error.

Implementation

Note: This tutorial is intended for viewing only. Any changes you might make within this tutorial cannot be transferred to your projects will not be saved when you upgrade Commander.

As you create steps for your projects, you can duplicate these tutorial concepts. On the New Steps page, review the Advanced section. Notice the Error Handling and Always Run Steps options.

- For Error Handling, choose the option that best suits your purpose.
- For the Always Run Step option, select the checkbox to ensure this step will always run.

Related information

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Storing and retrieving properties in a job

When a job is running, it is often necessary to share data between steps. Because the data being shared is specific to the running procedure instance, the job, the data is more appropriately stored at the job level rather than the procedure level. Use the techniques illustrated in this tutorial to safely pass data between steps in a job.

This tutorial shows you how to create, set, and retrieve properties stored on a job.

An explanation of what you see:

This procedure implements 2 steps.

- Write data to a property attached to the running job:
 "Step 1 - Create a property attached to the job" calls `ectool` to create a property name `property1`.
 - The path to the property being set is `"/jobs/${myJob}/property1`.
 - `${myJob}` is a shortcut to the running job name and can be used while the job is running.
 If the job number was 1000, the path `"/jobs/${myJob}/property1"` might expand to `"/jobs/job-1000/property1"`.
- Read data from the property created in Step 1:
 "Step 2 - Read the property attached to the job by Step 1" - this step could read step values stored on a job, running as part of the job, or could be executed using a simple property expansion.
 - In this case, `property1` was set in Step 1 and is referenced in Step 2 using the `${property1}` notation.
 - Two other ways to reference a property (shown by example in the Action column):
 - Using `/${myJob/property1}`
 - Using inline JavaScript with the notation `$/javascript myJob.property1]`

Click **Run** to run this sample procedure and see the resulting job on the Job Details page.

- After the job completes, click the icon in the Log column for Step 2 to view three lines each showing the data (the property value) that was stored in the property that was retrieved.

Implementation

If you need to retrieve properties, adapt these step concepts to your procedures.

- For help getting started, click on a step name in the tutorial procedure to go to the corresponding Edit Step page.
- You can reuse the text supplied in the Command(s) box, changing the "property1" name and other values to those more meaningful to your project/procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

[Procedure - create new or edit existing procedure](#) - help topic

[Step - create new or edit existing step](#) - help topic

Every ElectricCommander web page has a Help topic containing information about functionality and options available on that page. Click the **Help** link in the upper-right corner of any Commander page to access page help.

Working with properties stored in a procedure

Using Commander properties is one of the keys to robust yet flexible process automation. Configuration and historical data persistence are common tasks in process automation.

This tutorial shows common ways to read and write properties on a procedure.

Note: For this tutorial, familiarity with creating procedures and steps is assumed. See the "Related information" links below if you would like to review these topics.

An explanation of what you see—this procedure implements the following functionality:

- This procedure implements 5 steps. Each step name is a description of its action.
 - Click on a step name to go to the corresponding Edit Step page to see how the step was created. This page also contains text in the Description text box to describe what you see in the Action column on the Procedure Details page. The full text of what appears in the Action column is provided in the Command text box.
 - In the Custom Procedure Properties table, notice 2 properties were created: `property1` and `property2`
- There are numerous ways to work with stored properties. ***Each of the 5 steps described below illustrates a different method.***
 - Read data from `property1` using the *property substitution* notation (syntax):
The property value is read in this step using the property substitution reference `$/myProcedure/property1`.
 - Read data from `property1` using inline JavaScript:
The property name `property1` is read from the procedure level using this JavaScript notation: `$/javascript myProcedure.property1`.
 - Perform a calculation using data from `property 2` and inline JavaScript:
In this example, the value of `property2` is multiplied by 5 using this JavaScript notation: `$/javascript myProcedure.property2 * 5`.
 - Read data from a `property1` using the Commander Perl API:
Perl code is used to call the Commander API to retrieve the value of `property1`.
The `getProperty` API returns an object so the property value must be retrieved after the `getProperty` call, using the `findnodes` method for the returned object.

```
#read the property
my $propertylnode = $ec->getProperty("/myProcedure/property1");
#retrieve the property value from the object returned to $propertylnode
my $property1 = $propertylnode->findnodes("//value")->string_value();
```

- Read data from `property1` using ectool:
ectool code is used to retrieve the value of `property1`:

```
ectool getProperty "/projects/$/myProject/projectName]/procedures/Working
with properties stored in a procedure/property1"
```

Click **Run** to run this sample procedure and see the results on the Job Details page.

On the Job Details page, click the icon in the Log column to see the value read from the property.

Implementation

If you need to retrieve properties stored in a procedure, adapt any one of these step concepts to your procedures.

You can reuse the text supplied in the Command(s) box (from any Edit Step page in this tutorial), changing names and other values to those more meaningful to your project/procedure.

Note: Any changes you make within this tutorial will not be saved when you upgrade Commander.

Related information

[Procedure - create new or edit existing procedure](#) - help topic

[Step - create new or edit existing step](#) - help topic

[Properties](#) - help topic

[Using ectool and the Commander API](#) - help topic

[API commands](#) - help topic