API Specification Document

Rev 2.0

`

Table of Contents

Copyright Statement

CenterScape API Specification Rev. 2.0

RF Code, Inc.

9229 Waterford Centre Blvd. Suite 500

Austin, TX 78758 www.rfcode.com

1. Introduction

This specification is intended to provide the details needed for third-party software and driver developers to successfully produce code that integrates with RF Code CenterScape, version 2026.1 or later, produced by RF Code, Inc.

The interfaces provided are intended to expose all available features of the product in a supported and consistent fashion, allowing applications to be developed that will require little or no changes to support future versions of the CenterScape product. That said, RF Code does reserve the right to change any or all these interfaces in future product releases.

1.1. HTTP-based API Interface

The HTTP-based API interface for CenterScape is the supported interface for application implementation, as most programming languages and environments support efficient issuing of HTTP- based requests and parsing of the responses. In addition, most corporate networking systems support a variety of mechanisms for managing HTTP-based traffic through firewalls, proxies, and other technologies. HTTP-based APIs can easily be used by other web server-based applications and web browser-based applications (such as those implemented using JavaScript in HTML documents).

The CenterScape server follows a representational state transfer (REST) model for most of the APIs. The following table maps the HTTP method to the supported API operation:

HTTP Method	API Operation
POST	Create/Update
GET	Read
PUT	Update/Replace
DELETE	Delete

The HTTP methods map one-to-one to the relational database CRUD operations.

The REST model is resource-centric where the resources being operated on are specified in the URL to the right of the hostname. All CenterScape APIs are declared under a command URL path:

http://<server-hostname>:6580/api

For example, to request a specific CenterScape entity, you might construct the following HTTP

request:

GET http://<server-hostname>:6580/api/entity/$tAssetType_fb34e50

The GET HTTP command verb is used to retrieve information. The response that is returned is encoded in either a Javascript Object Notation (JSON) or Comma Separated Value (CSV) formatted response.

1.2. HTTP Authentication

CenterScape server access requires an authenticated user ID and password. The CenterScape server will accept either Form-based or Basic authentication to authenticate a user.

Since neither Form-based or Basic authentication is considered secure, the CenterScape server should be configured to use HTTPS if there is a concern the user ID or password can be intercepted.

1.3. Standard Output Formatting

Command output is encoded consistently for the two output formatting types, although the content of each command is specific to that command.

For JSON-encoded output, each command will return a valid JSON object or array. If an object is returned (JSON objects are enclosed in curly brackets “{ }”), it will contain one or more comma-separated attribute-value pairs. If an array is returned (JSON arrays are enclosed in square brackets “[ ]”), it will contain zero or more JSON objects. Any application processing the JSON-formatted output must check if the returned entity is an object or array.

For CSV-encoded output, the first line will be a header line containing the object identifiers and attributes that are being returned in the response set. Following the header line, there is a line for each entity that is returned in the response set.

1.4. Error Reporting

If an API returns a HTTP status code other than 200 (OK), a single JSON-encoded status object is returned. An example of a status object indicating a required field is missing a value would be:

{

"success" : false

, "errors" : [

{

"code" : "AttributeRequired"

, "message" : "Required value is missing for Entity Root"

, "field" : "$aName"

}

]

}

- ***Success*** is an attribute indicating the overall status of the request.
- ***Errors*** is a JSON array containing the one or more JSON objects indicating the error con- dition(s).
- ***Message*** is a human-readable error message indicating the error condition.
- ***Field*** is the ID of the attribute which has an error condition.

1.5. Macros

Macros are variables specified during the configuration of some attributes and which are replaced with actual values when the system sends or displays the attribute value. Macros can be used within:

- Email action messages
- Directory paths and filenames for various actions
- The titles of dashboard widgets

Macros are inserted by specifying the macro name within the text value of an attribute, prefaced with a dollar sign and enclosed between curly brackets. An example of an email event action configured to include the time that the event occurred would have the following value for the event action message:

```
Event Time: ${TIME}
```

Using this value, if an event occurs at 12:30:15 p.m. and the event is configured to execute the email action, then the email message sent when the event is triggered would be:

```
Event Time: 12:30:15 pm
```

Since alerts and events are generated from a source entity, and since the attributes that an entity has associated with it are definable by the system administrator, macros provide a way to specify that a specific attribute from an alert or event source should be inserted. This is done by using the macro name "SOURCE." followed by the ID of the attribute. The following example macro would insert an entity's description:

${SOURCE.$aDescription}

Macro	Available for Alerts	Available for Events	Available for report
TIMESTAMP	✓	✓	✓
DATE	✓	✓	✓
YEAR	✓	✓	✓
MONTH	✓	✓	✓
DAY	✓	✓	✓

TIME	✓	✓	✓
HOUR	✓	✓	✓
MINUTE	✓	✓	✓
SECOND	✓	✓	✓
MILLISECOND	✓	✓	✓
TIMEZONE_OFFSET	✓	✓	✓
ID	✓	✓	✓
SOURCE_ID	✓	✓	
NAME	✗	✗	✓
RESOLVE_TIME	✓	✗	✗
TYPE	✗	✗	✓
SEVERITY	✓	✗	✗
DESCRIPTION	✓	✓	✗
TRIGGER_NAME	✗	✓	✗
TRIGGER_TYPE	✗	✓	✗
TRIGGER_ATTRIBUTE1_ID	✗	✓	✗
TRIGGER_ATTRIBUTE1_NAME	✗	✓	✗
TRIGGER_OPERATOR1	✗	✓	✗
JOB_ID	✗	✗	✓
JOB_NAME	✗	✗	✓
JOB_START_TIME	✗	✗	✓
JOB_STOP_TIME	✗	✗	✓
START_TIME	✓	✗	✗

STATE	✓	✗	✗
THRESHOLD_NAME	✓	✗	✗
THRESHOLD_TYPE	✓	✗	✗
URL	✓	✗	✗
THRESHOLD_ATTRIBUTE1_ID	✓	✗	✗
THRESHOLD_ATTRIBUTE1_NAME	✓	✗	✗
THRESHOLD_OPERATOR1	✓	✗	✗
THRESHOLD_VALUE1	✓	✗	✗
THRESHOLD_ATTRIBUTE2_ID	✓	✗	✗
THRESHOLD_ATTRIBUTE2_NAME	✓	✗	✗
THRESHOLD_OPERATOR2	✓	✗	✗
THRESHOLD_VALUE2	✓	✗	✗
THRESHOLD_ATTRIBUTE3_ID	✓	✗	✗
THRESHOLD_ATTRIBUTE3_NAME	✓	✗	✗
THRESHOLD_ATTRIBUTE4_ID	✓	✗	✗
THRESHOLD_ATTRIBUTE4_NAME	✓	✗	✗
THRESHOLD_OPERATOR4	✓	✗	✗
THRESHOLD_VALUE4	✓	✗	✗
THRESHOLD_ATTRIBUTE5_ID	✓	✗	✗
THRESHOLD_ATTRIBUTE5_NAME	✓	✗	✗
THRESHOLD_OPERATOR5	✓	✗	✗
THRESHOLD_VALUE5	✓	✗	✗
TRIGGER_VALUE1	✗	✓	✗

TRIGGER_ATTRIBUTE2_ID	✗	✓	✗
TRIGGER_ATTRIBUTE2_NAME	✗	✓	✗
TRIGGER_OPERATOR2	✗	✓	✗
TRIGGER_VALUE2	✗	✓	✗
TRIGGER_ATTRIBUTE3_ID	✗	✓	✗
TRIGGER_ATTRIBUTE3_NAME	✗	✓	✗
TRIGGER_OPERATOR3	✗	✓	✗
TRIGGER_VALUE3	✗	✓	✗
FILTER_LOCATION	✗	✓	✓
FILTER_TYPE	✗	✓	✓
TYPE_ID	✗	✗	✓
SOURCE.attribute	✓	✓	✗
SOURCE_TRIGGER_VALUE 1	✗	✓	✗
SOURCE_TRIGGER_VALUE2	✗	✓	✗
SOURCE_TRIGGER_VALUE3	✗	✓	✗
TRIGGER_ID	✗	✓	✗
TRIGGER_TIME	✗	✓	✗
SOURCE_THRESHOLD_ ADAPTIVE_VALUE	✓	✗	✗
SOURCE_THRESHOLD_VALUE1	✓	✗	✗
SOURCE_THRESHOLD_VALUE2	✓	✗	✗
SOURCE_THRESHOLD_VALUE3	✓	✗	✗
SOURCE_THRESHOLD_VALUE4	✓	✗	✗
SOURCE_THRESHOLD_VALUE5	✓	✗	✗

THRESHOLD_ADAPTIVE_ ATTRIBUTE_ID	✓	✗	✗
THRESHOLD_ADAPTIVE_ ATTRIBUTE_NAME	✓	✗	✗
THRESHOLD_ADAPTIVE_ ATTRIBUTE_OPERATOR	✓	✗	✗
THRESHOLD_ADAPTIVE_VALUE	✓	✗	✗
THRESHOLD_ID	✓	✗	✗

NOTE: Unlike alerts and events, reports do not reference a source asset and therefore do not have any ${SOURCE.attribute_name} macros available.

1.6. Actions

Actions can be configured to execute in association with Alerts, Events, and Reports. These actions are designed to push information about the associated alert, event, or report to outside systems. Actions can be configured to send information using various protocols, including:

- For alerts, events, and reports:
  - email
  - ftp
  - sftp (SSH File Transfer)
  - http
  - https
- For alerts and events (but not reports)
  - Logging
  - SNMP v1 Traps
  - SNMP v3 Traps
  - SNMP v3 Informs
  - Commands to devices connected to a serial port

Serial device actions send the commands specified in the action. Email actions send the message specified in the action after doing proper text replacement of any macros specified in the message. The output of all other actions includes:

- The replaced values for all macros available to the action (see table in Macros above), with the exception of the following macros, which only output a partial date or time: DATE, YEAR, MONTH, DAY, TIME, HOUR, MINUTE, SECOND, MILLISECOND,
- TIMEZONE_OFFSET.
- Additional source attributes specified in the definition of the source alert or event (not applicable to reports)
- Other values for backwards compatibility of alert actions. All alert action name/value pairs from previous versions of HTTP and FTP alert actions are included and supported. Since some names do not match the macro name for the same value, the value is duplicated.

2. Data Model

The data model describes the major elements necessary to understand the functionality of RF

Code's CenterScape.

2.1. Object Types

The CenterScape data model centers around the definition of and interaction with a family of core object types. Each of these object types plays a role in supporting and defining the behavior and features of the CenterScape system, and many of the object types interact with one another to bring this about. Most of the object types require instances to have a globally unique object ID ("guid") – an alphanumeric, case-sensitive identifier string that must be unique and cannot be changed during an object's life cycle.

The four primary object types in the CenterScape data model are Entity Types, Attribute Classes, Attributes, and Entities. An Entity Type can be roughly described as what an object is; for example, an Entity Type could be a vehicle, or more specifically a car. An Attribute Class is used to describe the characteristics or properties of an Entity Type. An Entity is an object whose type is a reference to an existing Entity Type.

An instance of an Attribute is a name/value pair where the name refers to a defined Attribute Class and the value is the value of the attribute constrained to the Attribute Class type and any other constraints that may be specified in the declaration of the Attribute Class.

As you read this document, the terms Entity Type and Entity, Asset Type and Asset are used interchangeably. Entity Types and Entities are internal terms used to describe data model objects. Asset Types and Assets are customer terms used to describe the same data model objects.

2.2. Entity Type

Entity Types represent the different types of Entity objects defined within CenterScape. Entity Types are used to define the population of attributes that may be present on Entities (instances of a type). Each Entity Type must have a globally unique object ID (guid).

Entity Types can be arranged in a hierarchy by allowing one Entity Type to be specified as the "parent" of other Entity Types. Entity Type attributes inherit from parent to child, so setting the value of the City attribute of a parent Entity Type will cause that attribute value to be inherited by all of its children (and any descendants), unless those Entity Types provide their own value for the City attribute. A child Entity Type may override any of its parent Entity Type Attribute objects. A child may not, however, remove an Entity Type Attribute its parent defines.

Entity Type Properties	
class	Describes the class of object. For entity types the value is “entity_type”.
guid	The globally unique ID for this entity type.
parent	The guid of the parent entity type.
name	Presentation label.
descprtion	Describes this entity type in detail.
deletable	Describes whether a user can delete this entity type. Entity Type objects defined by the CenterScape server and not a user are marked as non-deletable.
attributes	An array of objects that contain the attribute classes which are used when creating instances of an entity type.

2.3. Entity

Entity objects provide the main interface for interacting with CenterScape. An Entity object represents a single instance of an asset or tag with one or more associated properties or attributes. Each Entity object has a single well-known attribute named Type which represents the base type of the Entity. The type of the Entity corresponds to an existing Entity Type object.

An asset has zero or more associated Attribute objects. Each Attribute corresponds to a single property of the Entity. The available attributes of the entity are defined by the entity’s Entity Type objects.

2.4. Attribute

An Attribute represents a single property of an asset. Each Attribute has an associated value. The data type of the Attribute’s value is defined within the Attribute’s associated Attribute Class object. Each attribute may store history depending on the *History Recorded* property of the Attribute Class.

2.5. Attribute Class

An Attribute Class represents the definition of an Attribute. In essence, an Attribute Class defines what the value of an asset property is allowed to be. Each Attribute Class must have a unique object ID. An Attribute Class has the following properties:

Attribute Class Properties		
Name	Description	Default Value(if none specified)
class	Describes the class of object. For attribute classes the value is “attribute_class”.	required
guid	The globally unique ID for this attribute class.	required
name	Presentation label.	required
description	Describes this attribute class in detail.	optional
deletable	Describes whether or not a user can delete this attribute class. Attribute Class objects defined by the CenterScape server and not a user are marked as non-deletable.	true
retired	If true, this attribute class is no longer accessible for editing. History can still be viewed. Updates to attributes of this class no longer occur.	false
history	If true, all changes to an attribute of this class are recordedin the history. If false, only the current value is stored.	false
values	Used only by string, string-list and enum Attribute Classes. An array which contains the list of values allowed for this attribute class. For the num type, each value corresponds to a specific enumerated value.	optional
inherit_attributes	If this attribute class’s type is type-ref, then entities will inherit the value of the attributes defined on the entity type.	optional
constraints	One or more constraints on the value of attributes of this class. Double and long data types can be constrained by a minimum or maximum value. A string, password, or stringlist can be constrained using a regular expression. A typeref, typeref-list, entityref and entityref-list values can be constrained by an Entity Type. For example, a user may constrain a location attribute to only values within the “Texas” Entity Type hierarchy.	optional
encoding	The type of encoding for a password attribute class. Blowfish and SHA-512 are supported.	required for some attributes of type “password”
type	The data type for values of this attribute class. See the table below for the list of data types	required
use	One of the following strings: info – Set by CenterScape and not a user. Used for informational values, such as the version of an application, which are visible to a user.	optional

	hidden – Set by CenterScape and not a user. These values are not shown to a user. CenterScape uses hidden values to associated metadata required to interact with an entity. For example, associating the tag type of a tag with a tag entity. config-view – CenterScape or a user may modify this attribute. All user created attributes are set to “config”. status – Set by CenterScape and not a user. Used for status values set by CenterScape which are visible to a user. For example, the motion value of a tag is determined by the CenterScape and not a user.	
$zName	Used only by Zone Manager. The $zName is an alias which points to Zone Manager’s name for this attribute. For example, in CenterScape the attribute may have the guid of “$aHost” while in Zone Manager the attribute is called host. In this case, the $zName value is “host”.	optional

The following table lists Attribute Class data types:

Attribute Clas Data Types	
bool	A true/false value (boolean)
date	A single date, such as January 31, 2026
double	8 bytes IEEE 754. Covers a range from 4.94065645841246544e-324d to 1.79769313486231570e+308d
entityref	A reference to an entity object. This is a simple association. The entity with this attribute does not inherit its entity referenced attributes
entityref-list	A list of references to entity objects. This is a simple association. The entity with this attribute does not inherit its entity referenced attributes
enum	An enumeration. The value of an attribute of this type is one of the values listed in the attribute class’s values property
long	8 bytes signed (two’s complement). Ranges from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
map	A hash table of values. A map must contain __mapkeytype and __mapvaluetype entries which contain the data type of the hash table key and value. Currently, map values may only be defined by the system and not a user.
password	Holds a password value. The encoding used is specified by the attribute class’s encoding property.
label set	Custom-defined String Lists used with Static Inheritance Mapping to allow values to be combined rather than overwritten. During set operations, any string with a leading - character is removed from a Label Set.
string	a string
string list	a list of strings
timestamp	A single date/time accurate to seconds and expressed in GMT. Note: The CenterScape web interface will adjust the value of a timestamp value by the browser’s time zone, and offset from GMT.

	No adjustment is made to a timestamp value if the value is being updated via an CenterScape API.

2.6. Entity Type Attribute

Once an Entity Type and an Attribute Class have been declared, a user associates an Attribute Class to an Entity Type. To do this, we use another CenterScape data model object called an Entity Type Attribute. The Entity Type Attribute is another object that describes how an Entity Type should display an Attribute Class, if the Attribute Class should have a default value, the order an Attribute Class should be presented relative to other Attribute Classes, and whether an Attribute Class's value is required.

The following table list the properties of an Entity Type Attribute:

Entity Type Attribute Properties	
class	Describes the class of object. For entity type attributes, the value is "entity_attribute".
guid	The globally unique ID of an existing attribute class.
required	If true, upon creation of an Entity a user must provide a value for this attribute.
isStatic	A static entity type attribute is an attribute which is an instance of the Entity Type and not the Entity. In this case, the Entity Type stores the current value and the history of the attribute. For example, a location Entity Type defines a static attribute named City with the value "Austin". All Entities which have a typeref attribute whose value is "location" then will also have an attribute named City with the value "Austin". A non-static entity type attribute makes an attribute available for the Entity to define. For example, a Server entity type may have the attribute RAM. In this case different servers have different amounts of RAM therefore the RAM attribute should be non-static.
deletable	Describes whether or not a user can delete this attribute class. Attribute Class objects defined by the CenterScape server and not a user are marked as nondeletable.
value	For static attributes this defines the value of the attribute. For non-static attributes this defines a default value. When a user creates an Entity the non-static attribute is populated with the default value which the user may change.
sortPriority	Defines the sort order for attributes on the Entity user interface (UI). Attributes with a lower sort priority are displayed first.
category	Attributes with the same category are shown on the Entity UI grouped together within a titled box. Category has a lower precedence than sort priority; as a result attributes with the same category may appear in two different titled boxes. For example, suppose an entity type contains attributes City with sort priority 100 and State with sort priority 300, both in the Location category, and a third attribute Host with sort priority 200 and category Network. In this case, the UI will

2.7. Custom Attribute Type

When an asset is created in CenterScape, a single asset type is selected. That asset type defines the properties associated with the asset. A custom attribute type allows the value of an attribute to contribute other attributes to the asset.

For example, let's say you have a laptop asset that is in Austin, Texas and when the asset is in Austin, Texas, you want its Building attribute set to "A-500". If that same laptop is in Palo Alto, California, you want its State Asset Code set to "C.1.33". The custom attribute type allows you to change the attributes that are associated with an asset by choosing one custom type versus another.

3. Data Schema

3.1. Attribute Classes

This section lists a subset of the built-in attribute classes and a description of their use. The attributes listed are those that end-users can apply to their own asset types or that are automatically added to a customer's assets because of a tag being associated to an asset. By convention, built-in attribute class guids start with "$a" or "$z".

$aAssetDoorOpen

$aAssetDoorOpen Properties	
guid	$aAssetDoorOpen
type	bool
name	Door
history	True
use	Status
description	Indicates if the door switch mounted on the asset is reporting a door opened condition

$aAssetTag

$aAssetTag Properties	
guid	$aAssetTag
type	entityref-list
name	Asset Tag
history	True
use	Config-view
Constraints	"typeref" : "$tTag"

	Value of this attribute must be constrained to references to entities whose type is $tTag or inherits from $tTag.
description	This attribute contains the reference to the tag with which an asset is associated. Even though this attribute is a list, CenterScape does not support associating more than one tag to an asset.

$aAssetHumidity

$aAssetHumidity Properties	
guid	$aAssetHumidity
type	Double
name	Humidity
history	True
use	Config-view
printf	%.1 f
Units	rh
description	Reports the humidity at the asset (raw value in %RH)

$aAssetImage

$aAssetImage Properties	
guid	$aAssetImage
type	mimeref
name	Asset Image
history	False
use	Config-view
description	Value of this attribute is an image file associated with an asset

$aAssetLockClosed

$aAssetILockClosed Properties	
guid	$aAssetILockClosed
type	Bool

name	Lock Closed
history	True
use	Status
description	The asset's lock closed status as reported by the asset's tag

$aAssetLockCount

$aAssetlLockCount Properties	
guid	$aAssetILockCount
type	Long
name	Lock Count
history	True
use	Status
description	The asset's lock count as reported by the asset's tag

$aAssetLockOpen

$aAssetlLockOpen Properties	
guid	$aAssetILockOpen
type	Bool
name	Lock Open
history	True
use	Status
description	The asset's lock open status as reported by the asset's tag

$aAssetLowBattery

$aAssetLowBattery Properties	
guid	$aAssetLowBattery
type	Bool
name	Low Battery
history	True
use	Status
description	Indicates if the asset has detected a low battery condition as reported by the asset's tag

$aAssetMotion

$aAssetMotion Properties	
guid	$aAssetMotion
type	Bool
name	Motion
history	True
use	Status
description	The asset's motion status as reported by the asset's tag

$aAssetMotionCount

$aAssetMotionCount Properties	
guid	$aAssetMotionCount
type	Long
name	Motion Count
history	True
use	Status
description	The asset's motion count as reported by the asset's tag

$aAssetPanic

$aAssetPanic Properties	
guid	$aAssetPanic
type	Bool
name	Panic
history	True
use	Status
description	Reports if the panic switch has been activated on the asset's tag

$aAssetSensorLowBattery

$aAssetSensorLowBattery Properties	
guid	$aAssetSensorLowBattery

type	Bool
name	Sensor Low Battery
history	True
use	Status
description	Reports if the asset has detected a low batter condition for an attached sensor

$aAssetTamper

$aAssetTamper Properties	
guid	$aAssetTamper
type	Bool
name	Tamper
history	True
use	Status
description	Indicates if the asset has detected a tamper attempt as reports by the asset's tag

$aAssetTemperature

$aAssetTemperature Properties	
guid	$aAssetTemperature
type	Double
name	Tem
history	True
use	Status
Printf	% 1 f
Units	Celsius
description	Reports the temperature of the asset(raw value in degressC)

$aAssetUserPayload

$aAssetUserPayload Properties	
guid	$aAssetUserPayload
type	long
name	User Payload
history	True

use	Status
description	An asset's numeric payload as reported by the asset's tag

$aBoundAsset

$aBoundAsset Properties	
guid	$aBoundAsset
type	entityref
name	Associated Asset
history	True
use	hidden
constraints	"typeref" : "$tEntity" Value of this attribute must be constrained to references to entities whose type is $tEntity or inherits from $tEntity.
description	This attribute constrains the reference to the entity (asset) to which a tag is bound

$aCookieData

$aCookieData Properties	
guid	$aCookieData
type	map
name	Cookie Data
history	false
use	hidden
description	This attribute is used by the CenterScape web UI to contain user UI settings

$aDescription

$aDescription Properties	
guid	$aDescription
type	String
name	Description
history	True
use	Config-view
description	The value of this attribute is intended to contain a descriptive label for an asset

$aDetectedLocation

$aDetectedLocation Properties	
guid	$aDetectedLocation
type	typeref
name	Detected Location
history	true
use	status
description	The location of an asset based on the server's configured location rules

$aReservedLocation

$aReservedLocation Properties	
guid	$aReservedLocation
type	Typeref
name	Reserved Location
history	True
use	Status
description	The future location of an asset based on the user input via UI or API

$aEnabled

$aEnabled Properties	
guid	$aEnabled
type	Bool
name	Enabled
history	True
use	Config-view
description	The value of this attribute is intended to control the enable/disable stat of an object

$aLocation

$aLocation Properties

guid	$aLocation
type	typeref
name	Asset Location
history	True
use	Config-view
Constraints	"typeref" : "$tLocation" Values of this attribute must be references to a entity type that is a descendant of the $tLocation type.
Inherit_attributes	true The attribute values associated to the location typeref will be inherited by the asset.
description	The value of this attribute is the current declared location on an asset

$aName

$aName Properties	
guid	$aName
type	String
name	Name
history	True
use	Config-view
description	The attribute is used to contain the name of an asset. Values of this attribute must be unique for all objects defined in CenterScape. The value of this attribute can be changed. The CenterScape web UI uses this attribute to uniquely identify or select assets. All user-defined assets types contain this attribute.

$aOnline

$aOnline Properties	
guid	$aOnline
type	Bool
name	Online Status
history	True
use	Status
description	If the value of this attribute is true, the tag associated with the asset is currently observable from one or more readers

$aPassword

$aPassword Properties	
guid	$aPassword
type	Password
name	Password
history	True
use	config
Encoding	“SHA-512”
description	Password attributes used for resources that need user/password authentication

$aServiceDate

$aServiceDate Properties	
guid	$aServiceDate
type	Date
name	Service Date
history	True
use	Status
description	This attribute contains the creation date of an object

$aViewAttributes

$aViewAttributes Properties	
guid	$aViewAttributes
type	String-list
name	View Attributes
history	false
use	config
description	A list of zero of more attribute guids that define the attributes to be displayed when a view is selected in the CenterScape web UI

$aRetired

$aRetired Properties

guid	$aRetired
type	Bool
name	Retired
history	True
use	hidden
description	This attribute is used to set the retired status of an asset. If the value of this attribute is false, an asset is considered active. If the value is true, an asset is considered retired. This attribute is used throughout the CenterScape web UI when generating reports or viewing an asset's attributes. When retiring an asset through the CenterScape web UI, the user has the option of disassociating its asset tag

$aType

$aType Properties	
guid	$aType
type	Typeref
name	Asset Type
history	True
use	hidden
constraints	"typeref" : "$tRoot" Value of this attribute must be a reference to $tRoot or any descendant of $tRoot.
description	This attribute is the type of any object in the system. The value of this attribute cannot be changed after an object is created.

3.2. Entity Types

Entity types can be defined hierarchically, such that child types inherit from their parents. The CenterScape server defines and uses many built-in types to configure and monitor the state of the CenterScape server. CenterScape administrators are expected to create new entity types which model their customer's assets.

Most of the built-in entity types defined are only accessible by a user with administrator privileges. This section lists the entity types a customer will encounter when defining

new entity types used when modeling their own asset types. By convention, all built-in entity type guids start with “$t”. All built-in entity types have their *deletable* property set to false.

$tRoot

$tRoot Properties	
guid	$tRoot
partent	
name	Root
description	This entity type is the root node in the entity type hierarchy for both system- and customer-defined entity types. With the exception of custom attribute types, all entity types have this type as their ancestor.

$tEntity

$tEntity Properties	
guid	$tEntity
partent	$tRoot
name	Entity
description	Base entity type for all user and system objects
attributes	$aName Every instance of $tEntity (or ancestor of $tEntity) has the $aName attribute. Val-ues of the $aName attribute must be unique for any type which has the $aName attribute. This attribute is used in multiple places in the web UI to uniquely identify and select entity instances

$tUserEntity

$tUserEntity Properties	
guid	$tUserEntity
partent	$tEntity
name	User Entity
description	Paretn type for user-related ojects

$tAsset

$tAsset Properties	
guid	$tAsset
partent	$tUserEntity
name	Asset
description	Base entity type for all user-related ojects
attributes	$aDescription $aLocation

$tAssetAlert

$tAssetAlert Properties	
guid	$tAssetAlert
partent	$tAlert
name	Asset Alert
description	Root of the entity type tree for alerts that are created from thresholds applied to assets

$tUnassignedTag

$tUnssisgnedTag Properties	
guid	$tUnassignedTag
partent	$tUserEntity
name	Unassigned Tag
description	Entity type used when creating instances of tags which have been accepted by a user to associate to an asset
Attributes	$aLocation

3.3. Custom Attribute Types

There are several built-in custom attributes types defined by the CenterScape server's schema, the most notable of which is $tLocation. The $tLocation custom attribute type is the root of all user-defined locations.

$tLocation

$tLocation Properties	
guid	$tLocation
partent	
name	Location
description	This entity type is the root node for all user-created locations

$tUnknownLocation

$tUnknownLocation Properties	
guid	$tUnknownLocation
partent	$tLocation
name	Unknown Location
description	This is the entity use to designate the vale of the $aLocation attribute of an asset when the location of an asset cannot be determined using the currently defined location rules.

4. Putting It Together: An Example

To illustrate the concepts outlined in the data model and data schema sections, a small data schema is presented. The following example will use JSON as the representation format.

The sample schema will model IT equipment. A good place to start is to create a list of object types that you want to model. This will help you determine how you want to classify your asset types. For this example, we will limit the equipment list to Servers.

Now that we know what we want to model, we need to think about the information we want to track for each of the asset types we have. The list of information you want to track on each object type will be a good starting place for the attribute classes you need to create for modeling your assets.

The root of the customer's data model begins by creating one or more child asset types from the built- in asset type *Asset*. CenterScape already associates several built-in attribute classes with the Asset entity type: Name, Description, Asset Assigned Location.

In this example, we're going to use the default object type hierarchy:

- Inventory
- Equipment
    - Blade Chassis
    - Blade Server
    - Network Equipment
    - Peripheral
    - Server
    - Storage

The JSON representation for these entity types (excluding the built-in type “$tAsset”) is expressed as:

```
[
    {
        "class" : "entity_type",
        "guid" : "INVENTORY",
        "name" : "Inventory",
        "parent" : "$tAsset",
        "deletable" : true,
        "description" : "",
    },
    {
        "class" : "entity_type",
        "guid" : "EQUIPMENT",
        "name" : "Equipment",
        "parent" : "$tInventory",
        "deletable" : true,
        "description" : "",
    },
    {
        "class" : "entity_type",
```

```
        "guid" : "SERVER",
        "name" : "Server",
        "parent" : "EQUIPMENT",
        "deletable" : true, "description" : "",
    }
]
```

For all IT assets, we want to track the Purchase Terms (Purchase, Leaser, or Loaner), Purchase Date, Purchase Value, Manufacturer, and Model.

The JSON representation for these attributes classes can be expressed as:

```
[
    {
        "class" : "attribute_type",
        "guid" : "PURCHASE_TERMS",
        "type" : "string",
        "deletable" : true,
        "history" : true,
        "inherit_attributes" : false,
        "subtype" : "",
        "name" : "Purchase Terms",
        "description" : "",
        "values" : [ "Purchase", "Lease", "Loaner" ]
    },
    {
        "class" : "attribute_type",
        "guid" : "PURCHASE_DATE",
        "type" : "date",
        "deletable" : true,
```

"history" : true,

"inherit_attributes" : false,

"subtype" : "",

"name" : "Purchase Date",

"description" : "",

"values" : [ ]

},

{

"class" : "attribute_type", "guid" : "MODEL",

"type" : "string",

"deletable" : true,

"history" : true,

"inherit_attributes" : false,

"subtype" : "",

"name" : "Model",

"description" : "",

"values" : [ ]

},

{

"class" : "attribute_type",

"guid" : "MANUFACTURER",

"type" : "string", "deletable" : true, "history" : true, "inherit_attributes" : false,
"subtype" : "",

"name" : "Manufacturer", "description" : "",

"values" : ["Acer", "American Power Conversion", "Apple", "Asus", "BenQ",
"Brocade", "Brother", "Canon", "Cisco Systems", "Compaq", "Dell", "D-Link", "EMC
Corporation", "Epson", "Everex", "Extreme Networks", "Foundry Networks",
"Fujitsu", "Fujitsu Siemens", "Gateway", "Hewlett-Packard", "Hitachi", "IBM",
"Iomega", "Juniper Networks", "Konica Minolta", "Leibert", "Lenovo", "Lexmark",
"LG Electronics", "Linksys", "Logitech", "Maxtor", "NEC", "NCR","Nortel", "OKI",
"Olivetti", "Panasonic", "Philips", "Ricoh", "Samsung", "Seagate", "Sharp", "SMC

```
                Networks", "Sony", "Sun Microsystems", "Toshiba", "Unisys", "ViewSonic",
                "WatchGuard", "Westinghouse", "Xerox" ]
        },
        {
                "class" : "attribute_type",
                "guid" : "PURCHASE_VALUE",
                "type" : "double",
                "deletable" : true,
                "history" : true,
                "inherit_attributes" : false,
                "subtype" : "",
                "name" : "Purchase Value",
                "description" : "",
                "constraints" : {
                "min" : 1.0
            },
"values" : [ ]
    }
]
```


Let's examine a few of the attribute classes individually to illustrate some of the capabilities of the CenterScape data model.

- **Purchase Terms:** In the attribute class *Purchase Terms*, we see the type of the attribute class is "string". The attribute class also defines the values as "["Purchase", "Lease", "Loaner"]". By specifying the *values* field in the attribute class, we are restricting the value of this field to be one of the values listed in the values array. A value for this attribute other than one of these strings will be flagged by the server as invalid.
- **Purchase Date:** This represents the date an IT asset was purchased.
- **Model:** The model of the IT asset. Any string value is valid for this attribute class.
- **Manufacturer:** Like Purchase Terms, this attribute class is a string whose values are limited to those specified in the *values* array.

- **Purchase Value:** The purchase price of the IT asset. The minimum value for values of this attribute class cannot be lower than 1.0; otherwise the CenterScape server will indicate the value is invalid.

Up to this point we have three entity types and five new attribute classes. The attribute classes have been defined but they have not been assigned to any asset type. To assign an attribute class to an entity type, we must use entity type attributes.

In the following example, Purchase Terms, Purchase Date, Manufacturer, and Purchase Value have all been added to Equipment.

```
[
        {
                "class" : "entity_type",
                "guid" : "EQUIPMENT",
                "name" : "Equipment",
                "parent" : "$tInventory",
                "deletable" : true,
                 "description" : "",
                 "attributes" : [
        {
                "guid" : "PURCHASE_TERMS",
                "class" : "entity_attribute",
                "deletable" : true,
                "required" : false,
                "isStatic" : false,
                "category" : "Basic Information",
                "sortPriority" : 50
        },
        {
                "guid" : "PURCHASE_DATE",
                "class" : "entity_attribute",
```

```
            "deletable" : true,
            "required" : false,
            "isStatic" : false,
            "category" : "Basic Information",
            "sortPriority" : 60
        },
        {
            "guid" : "MANUFACTURER",
            "class" : "entity_attribute",
            "deletable" : true,
            "required" : false,
            "isStatic" : false,
            "category" : "Basic Information",
            "sortPriority" : 70
        },
        {
            "guid" : "MODEL",
            "class" : "entity_attribute",
            "deletable" : true,
            "required" : false,
            "isStatic" : false,
            "category" : "Basic Information",
            "sortPriority" : 80
        },
        {
            "guid" : "PURCHASE_VALUE",
            "class" : "entity_attribute",
            "deletable" : true,
```

```
            "required" : false,
            "isStatic" : false,
            "category" : "Basic Information",
            "sortPriority" : 65
        }
]
```

As you can see, the declaration of the entity type Equipment has grown considerably. The difference is the addition of the entity type attribute *Attributes*. *Attributes* is an array of all attribute classes that have been added to the definition of the entity type. Each entry in this array is an entity type attribute.

The entity type attribute *sortPriority* determines the order of an attribute independent of entity type attribute *Category*.

If we want to track additional information that is specific to one of the child types of Equipment, we would update their entity type declarations to include the additional entity type attributes. For this example, we will introduce one additional attribute class for Servers. The additional Server Attribute Class is:

```
    {
            "class" : "attribute_type",
            "guid" : "SERVER_FORM_FACTOR",
            "type" : "string",
            "deletable" : true,
            "history" : true,
            "inherit_attributes" : false,
            "subtype" : "",
            "name" : "Server Form Factor",
            "description" : "",
            "values" : [ "Rackmount", "Blade", "Micro" ]
    }
```

To add this attribute to the “Server” entity type, the new “Server” entity type declaration would be:

```
[
        {
                "class" : "entity_type",
                "guid" : "SERVER",
                "name" : "Server",
                "parent" : "EQUIPMENT",
                "deletable" : true,
                "description" : "",
                "attributes" :
        [
                {
                        "guid" : "SERVER_FORM_FACTOR",
                        "class" : "entity_attribute",
                        "deletable" : true,
                        "required" : false,
                        "isStatic" : false,
                        "category" : "Server Information",
                        "sortPriority" : 1400,
                        "value" : "Rackmount"
                        }
                ]
        }
]
```


The new attribute is added in a new category: Server Information. Since Server inherits from Equipment, it has all the attributes of an Equipment asset plus the Server Form Factor attribute. Also, the *Value* field has been added to the entity type attribute. The value Rackmount will be used as the default value for the Server Form Factor attribute.

This small sample has addressed attribute classes, entity types and entity type attributes. One powerful feature of the CenterScape data model which has not been demonstrated is the use of custom attribute types. A custom attribute type looks identical to an entity type

except the root entity type of a custom attribute type does not have a parent.

For this example, let's say we want to track the department to which an IT Asset is assigned. We could easily create a new string attribute class whose values are all the valid department names that exist in your organization. As you dig a bit deeper, perhaps you realize when an IT Asset is assigned to a department, there is additional information that each department maintains for its IT Equipment and that list of attributes each department maintains is different from department to department. A custom attribute type is the perfect choice for modeling this information.

For this example, the custom type hierarchy is:

- Department
    - Engineering
    - Finance

The JSON representation for these entity types would be expressed as:

```
[
        {
                "class" : "entity_type",
                "guid" : "DEPARTMENT",
                "name" : "Department",
                 "deletable" : true,
                "description" : ""
        },
        {
                "class" : "entity_type",
                "guid" : "FINANCE",
                "name" : "Finance",
                "parent" : "DEPARTMENT",
                "deletable" : true,
                 "description" : ""
        },
```

```
    {
        "class" : "entity_type",
        "guid" : "ENGINEERING",
        "name" : "Engineering",
        "parent" : "DEPARTMENT",
        "deletable" : true,
        "description" : ""
    }
]
```

As you can see the, the Custom Attribute Type declarations are identical to those declared for the object type hierarchy. Also, it is worth mentioning the Department entity type has no parent. We can now associate additional attributes to the Finance entity eype and Engineering entity type. The new attributes are *Manager, Contains Employee Data,* and *System Use*, expressed using the following JSON:

```
[
    {
        "class" : "attribute_type",
        "guid" : "MANAGER",
        "type" : "string",
        "deletable" : true,
        "history" : true,
        "inherit_attributes" : false,
        "name" : "Manager",
        "use" : "config-view",
        "description" : "",
        "values" : [ ]
    },
    {
        "class" : "attribute_type",
```

```
            "guid" : "CONTAINS_EMPLOYEE_DATA",
            "type" : "bool", "deletable" : true, "history" : true, "inherit_attributes" : false,
            "name" : "Contains Employee Data", "use" : "config-view",
            "description" : "",
            "values" : [ ]
        },
        {
            "class" : "attribute_type",
            "guid" : "SYSTEM_USE",
            "type" : "string",
            "deletable" : true,
            "history" : true,
            "inherit_attributes" : false,
            "restrictable" : false,
            "name" : "System Use",
            "use" : "config-view",
            "constraints" : {
            "values" : true
        },
         "values" : [ "Build", "Development", "Test", "Production" ]
         }
]
```


As before, we apply the attribute classes to the entity types. In this example, the Department entity type remains the same. Each child type of Department has new Attribute Classes added. The Manager attribute class has been associated with both Engineering and Finance. When the Manager attribute is applied to the Engineering entity type , its value is “Bob Murphy” and *isStatic* is true. This means the value is constant and cannot be overridden. When Manager is applied to Finance, its value is “Tom Kendle”. Again, it is declared to be static and its value cannot be overridden. The System Use attribute class is

associated with the Engineering entity type and the Contains Employee Data attribute class is associated with the Finance entity type. The updated entity type definitions are expressed using the following JSON:

```
[
        {
                "class" : "entity_type",
                "guid" : "DEPARTMENT",
                "name" : "Department",
                "deletable" : true,
                "description" : ""
        },
        {
                "class" : "entity_type",
                "guid" : "ENGINEERING",
                "name" : "Engineering", "
                parent" : "DEPARTMENT",
                "deletable" : true,
                "description" : "",
                "attributes" : [
                {
                        "guid" : "SYSTEM_USE",
                        "class" : "entity_attribute",
                        "deletable" : true,
                        "required" : false,
                        "isStatic" : false,
                        "category" : "Department",
                        "sortPriority" : 2010
                },
                {
```

```
                        "guid" : "MANAGER",
                        "class" : "entity_attribute",
                        "deletable" : true,
                        "required" : true,
                        "isStatic" : true,
                        "value" : "Bob Murphy",
                        "category" : "Department",
                        "sortPriority" : 2000
                }
        ]
},
{
        "class" : "entity_type",
        "guid" : "FINANCE",
        "name" : "Finance",
        "parent" : "DEPARTMENT",
        "deletable" : true,
        "description" : "",
        "attributes" : [
                {
                        "guid" : "CONTAINS_EMPLOYEE_DATA",
                        "class" : "entity_attribute",
                        "deletable" : true,
                        "required" : false,
                        "isStatic" : false,
                        "value" : false,
                        "category" : "Department",
```

```
                "sortPriority" : 2010
            },
            {
                "guid" : "MANAGER",
                "class" : "entity_attribute",
                "deletable" : true,
                "required" : true,
                "isStatic" : true,
                "value" : "Tom Kendle",
                "category" : "Department",
                "sortPriority" : 2000
                }
            ]
        }
]
```

The custom types have been created, and new attribute classes have been associated with these types. Now we need a way for associating the custom entity types to the entity type hierarchy. The first thing we need to do is create a new attribute class that references the custom type hierarchy. This new attribute is called Owning Department, represented by:

```
{
        "class" : "attribute_type",
        "guid" : "OWNING_DEPARTMENT",
        "type" : "typeref",
        "deletable" : true,
        "history" : true,
        "inherit_attributes" : true,
        "restrictable" : false,
        "name" : "Owning Department",
        "use" : "config-view",
```

```
        "constraints" : {
                "typeref" : "DEPARTMENT",
                "typeref_select" : "leaf"
         }
}
```

This attribute class's type is *typeref*. This means the value of this attribute must be a guid of an Entity Type. The *constraints* object is used to limit the branch or node in the entity type hierarchy for valid values of this attribute class. Now we apply this attribute just like any other to a node in the entity type hierarchy. In this case, we'll say all Servers should have an Owning Department so we add the attribute to the Server entity type, expressed as:

```
[{
        "class" : "entity_type",
        "guid" : "SERVER",
        "name" : "Server",
        "parent" : "EQUIPMENT",
        "deletable" : true,
        "restrictable" : false,
        "description" : "",
        "attributes" : [{
                "guid" : "$aMountType",
                "class" : "entity_attribute",
                "deletable" : true,
                "required" : true,
                "isStatic" : false,
                "value" : "MOUNT_TYPE_RACK_U_MOUNT",
                "category" : "Rack Configuration",
                "sortPriority" : 60
},{
                "guid" : "OWNING_DEPARTMENT",
```

```
                "class" : "entity_attribute",
                "deletable" : true,
                "required" : false,
                "isStatic" : false,
                 "category" : "Department",
                "sortPriority" : 2000
        } ]
} ]
```

5. Command Reference

The HTTP command URLs used to configure and monitor the CenterScape server are presented in this section. For each HTTP command, a functional description is provided for the four basic HTTP methods (GET, POST, PUT, and DELETE).

The minimum privilege required to execute a command is specified. If an attempt to execute a command is made with credentials that do not meet the minimum privilege, a HTTP status code of 401 is returned.

5.1. Attribute Class

COMMAND

<base-url>/attributeclass

DESCRIPTION

The attributeclass command is used to create, update, or delete attribute classes.

GET

Retrieves one or more attribute classes.

Command syntax and parameters:

- attributeclass
  - Retrieves all attributes classes.
- attributeclass/ <object-guid>
  - object-guid (required) – The guid of the attribute class to retrieve.
  - Retrieves the attribute class with the specified object guid.
- attributeclass?type=<object-type-guid>
  - type = object-type-guid (required) – The object type guid.

- Retrieves all attributes classes which are defined for the specified object type and all its inherited type(s).

Command output:

If the command is successful, the output is either a JSON-encoded array of the requested attribute classes or a single JSON object representing the requested attribute class.

Minimum privilege level required:

Asset Viewer

Curl example:

curl -u user:password
'http://<serverhostname>:6580/api/attributeclass/$aAssetMotion'

POST

Creates an attribute class.

Command syntax and parameters:

- attributeclass/ POST body contains a JSON-encoded attribute class object.

Command output:

If the command is successful, indicated by HTTP status code 201, no output is returned. If the command is unsuccessful, indicated by HTTP status code 400, a JSON- encoded errors object is returned.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X POST -H 'content-type: application/json' -u admin:admin -d @<post-body-

filename> 'http://<server-hostname>:6580/api/attributeclass'

Where <post-body-filename> is the name of the file containing the post body.

PUT

Updates an existing attribute class.

Command syntax and parameters:

- attributeclass/<object-guid>

  object-guid (required) – Attribute class object guid.

  Updates the attribute class with the specified object guid.

POST body contains a JSON-encoded attribute class object.

Command output:

If the command is successful, indicated by HTTP status code 201, no output is returned. If the command is unsuccessful, indicated by HTTP status code 400, a JSON encoded errors object is returned.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X PUT -H 'content-type: application/json' -u admin:admin -d @<post-body-filename> 'http://<server-hostname>:6580/api/attributeclass/object-guid'

Where <post-body-filename> is the name of the file containing the post body.

DELETE

Deletes an existing attribute class.

Command syntax and parameters:

attributeclass/<object-guid>

object-guid (required) – Attribute class object guid.

Deletes the attribute class with the specified object guid.

Command output:

If the command is successful, indicated by HTTP status code 200, no output is returned.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X DELETE -u admin:admin
'http://<serverhostname>:6580/api/attributeclass/object-guid'

5.2. Audit Log

COMMAND

<base-url>/audit

DESCRIPTION

The audit command is used to retrieve audit log messages.

GET

Retrieves audit log messages within a specified time range or all audit messages.

Command syntax and parameters:

- audit?from=<utc-start-time>&to=<utc-end-time>&start=<starting-index>&limit=<max-result-count>

*from (*optional) – UTC start time value
*to (*optional) – UTC end time value
*start (*optional) – The index of the first item in the result set from which to return results.
For example, if a audit query returns 100 items, specifying a start value of 5 will prune off the first 5 elements from the result set.
*limit (*optional) – The maximum number of log messages to return.

Command output:

If the command is successful, a JSON object is returned with the count of audit

messages and an array of audit messages which include the timestamp in milliseconds the audit message was logged, the user which initiated the action and the message indicating the change which occurred. The following is a sample response:


```
{
"count" : 2,
        "data" : [
        {
        "timestamp" : 1225303027379,
        "message" : "User Login - admin - from remote address 127.0.0.1",
        "user" : "admin"
        },
{
        "timestamp" : 1225303023679,
        "message" : "User Login - foo - from remote address 127.0.0.1",
        "user" : "foo"
        }
]
                }
```


Minimum privilege level required:

System Administrator

Curl example:

curl -u admin:admin http://<server-hostname>:6580/api/audit?start=2&limit=10

POST

N/A

PUT

N/A

DELETE

N/A

5.3. Entity

COMMAND

<base-url>/entity

DESCRIPTION

The audit command is used to retrieve audit log messages.

GET

Retrieves one or more entities.

Command syntax and parameters:

- entity/<object-guid>

  object-guid (required) - The guid of the object to retrieve.

- entity?type=<entity-type-guid>

  entity-type-guid (required) – The guid of an entity type. All instances of the specified type are retrieved.

Command output:

If the command is successful, an array of JSON objects or a single JSON object is returned.

Minimum privilege level required:

The minimum privilege level required is dependent on the type of the object being requested. In general, system object types require System Administrator privilege, whereas user asset types require the least privileged user Asset Viewer

Curl example:

curl -u viewer:viewer http://<server-hostname>:6580/api/entity/object-guid

Creates an entity of a specific type.

POST

Creates an entity of a specific type.

Command syntax and parameters:

- entity/

POST body contains the JSON-encoded object to create. The object is validated using the semantics of the type of entity to create and the attribute classes referenced by this type and its inherited types.

Command output:

If the command is successful, the response is the JSON-encoded object of the newly created object. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object creation failed.

Minimum privilege level required:

The minimum privilege level required is dependent on the type of the object being created. In general, creation of system objects requires System Administrator privilege, whereas creation of user asset objects requires the least privileged user Asset Viewer.

Curl example:

curl -i -X PUT -H 'content-type: application/json' -u admin:admin -d @<post-bodyfilename> http://<server-hostname>:6580/api/entity/object-guid

Where <post-body-filename> is the name of the file containing the post body.

PUT

Updates an entity of a specific type.

Command syntax and parameters:

- entity/<object-guid>

Post body contains a JSON-encoded object to update. Attributes are removed from an entity by setting their attribute value to 'null'.

You can use this command to update one or more attributes on one or more entities using several parameters which allow you to specify the set of entities to update.

filter (required) – Boolean set to true to indicate a filter definition is included in the command parameters.

type (required) – The guid of the entity types to update.

attribute (optional) – The guid of the attribute class to filter by.

operator (optional) – The filter operator to apply to the attribute and value.

value (optional) – The attribute value.

attribute2 (optional) – The guid of the second attribute class to filter by.

operator2 (optional) – The second filter operator to apply to the attribute and value.

value2 (optional) – The second attribute value.

Command output:

If the command is successful, the response is the JSON-encoded object of the newly updated object. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object creation failed.

Minimum privilege level required:

The minimum privilege level required is dependent on the type of the object being updated. In general, updating system objects requires System Administrator privilege, whereas updating user asset objects requires the least privileged user Asset Viewer.

Curl example:

curl -i -X PUT -H 'content-type: application/json' -u admin:admin -d

@<postbodyfilename>

http://<server-hostname>:6580/api/entity/object-guid

Where <post-body-filename> is the name of the file containing the post body.

curl -i -X PUT -H 'content-type: applicaton/json' -u admin:admin -d @ <post-bodyfilename>
http://<serverhostname>:6580/api/entity?filter=true&type=$tAsset&attribute=$aLocation&operator=eq&value=TEXAS''

The file contains one or more attributes to update which match the filter set. In this case, the update set would be all assets with the current location of TEXAS. The post body filename can contain one or more attributes. If there is an OPERATIONAL_STATE attribute, the post body filename is a JSON object that might look like:

{ OPERATIONAL _STATE : "Available" }

You can also use the filter update method to delete an attribute value for one or more assets by setting the value of an attribute to null in the JSON post body. For example, you can remove the operational state for all assets in TEXAS by using the following JSON in the post body:

{ OPERATIONAL_STATE : null }

The JSON contained in the post body file can optionally contain another attribute $aGUIDSet. The value of this attribute is a JSON array of entity guids. Only entities contained in the guid set will be updated out of all the entities which match the filter criteria. The JSON might look like:

{ $aGUIDSet : "TEMPERATURE_HUMIDITY_335896177fe71f6b", "TEMPERATURE_HUMIDITY_4a209f006e59c1bb", "TEMPERATURE_HUMIDITY_529a4e111bc5b7f8", OPERATIONAL_STATE :”Installation Pending"}

DELETE

Deletes an entity

- entity/<object-guid>

object-guid (required) – The guid of the entity to delete.

Similar to the PUT method, you can use this command to delete one or more entities

which match a specific filter criteria.

type (required) – The guid of the entity types to update.

attribute (optional) – The guid of the attribute class to filter by.

operator (optional) – The filter operator to apply to the attribute and value.

value (optional) – The attribute value.

attribute2 (optional) – The guid of the second attribute class to filter by.

operator2 (optional) – The second filter operator to apply to the attribute

and value.

value2 (optional) – The second attribute value.

5.4. Configuration Import

COMMAND

<base-url>/configimport

DESCRIPTION

The configimport command is used to create or update existing asset entities, entity types or attribute classes in bulk. It can also be used to import a custom data schema.

GET

N/A

POST

The post is either a multi-part form upload or contains in the body a JSON-encoded

object which indicates the schema to load.

Command syntax and parameters:

- configimport

This command supports two different methods for creating or updating entities. The first format is a multi-part form upload. The format of the data is either CSV or JSON where the mime type of a CSV part should be “text/csv” and “application/json” for a JSON part. If the file being uploaded has an extension that ends with “.csv”, it will be interpreted as a CSV file regardless of the part’s content type.

The JSON representation for a single entity type is expressed as:

{

"class" : "entity_type",

"guid" : "INVENTORY",

"type” : ___;

"name" : "Inventory",

Specification 66

"parent" : "$tAsset",

"deletable" : true,

"description" : "",

}

The second method is used for importing a built-in data schema. The request is a normal POST where the post body is a JSON-encoded object that indicates which custom schema the CenterScape server should load. The format of a JSON object for a schema upload is:

{

" schema" : "schema name"

}

Command output:

The output of the command is a JSON-encoded status object indicating whether or how much of the data was imported out of the request. If it was not successful, the status object contains an array of error messages

indicating the reason or reasons why the request failed; if successful, provides a summary of what was imported. Example:

{

"success" : true,

"jobid" : 1079961581,

"messages" :

[

{ "code" : "import.successful", "message" : "Configuration file successfullyuploaded.\n\n"

}

,

{ "code" : "import.entitycount", "message" : "Number of assets imported: 1"

}

Minimum privilege level required:

The minimum privilege level required is dependent on the type of the objects being imported. Since the file of objects being imported may contain assets of different types, the requester must have the privilege level necessary to create or update all assets contained in the file being uploaded, otherwise the request will fail.

Curl example:

curl -X POST -u admin:admin -F 'filecontent=@example-import.csv' http://<serverhostname>:

6580/api/configimport

Where <example-import.csv> is the name of the file containing the post body.

PUT

N/A

DELETE

N/A

5.5. Configuration Export

COMMAND

<base-url>/configexport

DESCRIPTION

The *configexport* command is used to export existing entities, entity types, or attribute classes.

GET

This command can be used to export a collection of CenterScape objects (either system objects or customer's assets).

Command syntax and parameters:

- configexport

The configexport command supports a variety of methods to export information from the CenterScape server. The first method allows System Administrators to export the user asset schema (Asset Type, Attribute Classes, and Custom Attribute Types). The configexport command supports three ways of exporting data: schema export, export by filter, export by guid.

Schema export allows you to back up your customer asset schema by exporting all the customer's assets types, attribute classes, and custom attribute types.

Export by filter allows you to specify a filter condition an asset must match for it to be exported.

Export by guid allows you to explicitly specify the guids of the assets to export.

The following are the supported parameters for this command:

General parameters:

format = <"JSON"|"CSV"> Set the response format.

Schema export parameters:

schema = <true> Indicates the customer's asset schema should be exported.

Filter export parameters:

To export by filter, you would use the same parameters as those for the api/filter command.

filter = <true> Indicates the *configexport* command is exporting the assets that match those of the filter defined in the request body.

type = <type-guid> (required when exporting by filter) Specifies the filter asset types.

location = <location-type-guid> (optional) Specifies the filter location of an asset.

attribute = <attribute-guid> (optional) Specifies the guid of the attribute used in the filter condition.

operator = <"eq" | "ne" | "gt" | "lt" | "ge" | "le", "undef"| "def", "contains" | "containselement" | "startswith" | "in" | "inlist"> (optional) Specifies the operator used when comparing the value of the filter attribute.

value = <attribute comparison value> (optional) Specifies the value to compare in the filter condition.

attribute2 = <attribute-guid> (optional) Specifies the guid of the second attribute used in the filter condition.

operator2 = <“eq” | “ne” | “gt” | “lt” | “ge” | “le” | “undef” | “def” | “contains” | “containselement” | “startswith” | “in” | “inlist” | > (optional) Specifies the second operator used when comparing the value of the filter attribute.

value2 = <second attribute comparison value> (optional) Specifies the second value to compare in the filter condition.

attributeset = (optional) <attribute-guid | “*”> (optional) Specifies the list of attributes that should be returned for the assets matching the filter. The parameter is repeated for each attribute that is requested. If the value of this parameter is “*”, then all attributes for the assets matching the filter are returned in the result.

attributesort = <attribute-guid> (optional) Specifies the attribute used to sort the filter results

sortorder = <“asc” | “desc” | “none”> (optional) Specifies the sort order applied to the attributesort parameter.

showretired = <“yes” | “no” | “both”> (optional) Specifies if the result set should include retired assets. The default is “no”.

dateformat = <“iso8601”> (optional) Specifies if time stamps and date attributes should be formatted using the ISO 8601 format.

mimerefs = <“true” | “false”> (optional) Specifies if the result set should include

mime objects (image files) if assets have attributes referencing them.

prettyprint= <“true” | “false”> (optional) Specifies if the result set should use an attribute’s name, rather than <guid>, to represent an attribute.

resolverefs= <“true” | “false”> (optional) Specifies if the result set should use a custom attribute in the format of <attribute-guid>.$aName, rather than <attribute-guid>, and its value should be the name of an asset, rather than <entity-guid>.

Guid export parameters:

guid= <entity-guid> Specifies the guid of the asset to export. This parameter is repeated for every asset to export.

Schema export parameters:

schema= <“true” | “false”> Specifies if the result set returns the current custom schema.

Config export parameters:

type= <entity-type-guid> Specifies the type of entity (configuration) to export. This parameter is repeated for every entity type to export. <entity-type-guid> includes
$tSmtpServer, $tRanger, $tReader, $tTagGroup, $tSensorDefinition, $tLocation, $tRule, $tUser, $tGroup, $tLdapAuthenticationMethod, $tAssetLink, $tAssetAlertAction, $tAssetAlertThreshold, $tAssetTemplate, $tAttributeView, $tUserDashboard, $tFolder, $tMap, $tMapView, $tHotspot, $tHotSpotTarget, $tReport, $tReportAction, $tBirtTemplate, $tEventTrigger, $tEventTrigger, $tMonitor, $tStatPack, $tStatPolicy.

Command output:

The output of the command is a file containing the requested assets or schema to export. If the format of the request is “JSON,” the file contains a JSON array of matching requested objects. If the format is “CSV”, a comma-separated file is

returned.

Minimum privilege level required:

The minimum privilege level required is dependent on the type of the objects being exported. Since the file of objects being exported may contain assets of different types, the requester must have the privilege level necessary to view all the assets contained in the file being downloaded, otherwise the request will fail. If the request is to export the customer asset schema, System Administrator privilege is required.

Curl example:

curl -u admin:admin -i '<server-hostname>:6580/api/configexport?filter=true&type=$tAsset&dateformat=iso8601' (user-friendly date)

POST

Same as GET. Parameters must all be URL-encoded in the body of the post.

Curl example:

Export custom schema:

curl -u admin:admin -d 'schema=true' -i '<server-

hostname>:6580/api/configexport'

Export server configuration and smtp server setting:

curl -u admin:admin -d
'type=$tServerConfig&type=$tSmtpServer' "<server-hostname>:6580/api/configexport"

Export all assets with user-friendly date/time:

curl -u admin:admin -d 'filter=true&type=$tAsset
&dateformat=iso8601' -i '<server-
hostname>:6580/api/configexport'

Export Temperature/Humidity sensors with Rack Position Top and current temperature greater than 20 degrees Celsius:

curl -u admin:admin -d
'filter=true&type=TEMPERATURE_HUMIDITY&attribute=RACK_P
OSITION&-
operator=eq&value=Top&attribute2=$aAssetTemperature&oper
ator2=gt&value2=20' -i '<server-
hostname>:6580/api/configexport'

PUT

N/A

DELETE

N/A

5.6. Current User

COMMAND

<base-url>/currentuser

DESCRIPTION

The *currentuser* command is used to retrieve information, including authorized roles, for the currently authenticated browser user.

GET

The command returns a JSON-encoded object containing user information.

Command syntax and parameters:

current user

Command output:

The output of the command is a JSON-encoded object containing the current user information.
Minimum privilege level required:

Asset Viewer

Curl example:

curl -i -u admin:admin http://<server-hostname>:6580/api/currentuser

POST

N/A

PUT

N/A

DELETE

N/A

5.7. Change Password

COMMAND

<base-url>/changepassword

DESCRIPTION

The *changepassword* command is used to allow unprivileged users the ability to modify their own password. .

GET

N/A

POST

Updates the password for the currently authenticated user.

Command syntax and parameters:

- changepassword

The body should contain a URL-encoded form the with the following parameters: *userName* (required) – The name of the user for which to change the password *currentPassword* (required) – The user's current password. *newPassword* (required) – The new password for the specified user.

Command output:

If the command is successful, indicated by HTTP status code 200, no output is returned.

Minimum privilege level required:

Asset Viewer

Curl example:

```
curl -i -X POST -u user:password -
d"userName=user&currentPassword=password&new
Password=newpassword" http://<server-
hostname>:6580/api/changepassword
```

PUT

N/A

DELETE

N/A

5.8. Database Configuration

COMMAND

<base-url>/databaseconfiguration

DESCRIPTION

The databaseconfiguration command is used to create, update or retrieve the configuration to connect to a database server.

GET

Retrieves the current database configuration.

Command syntax and parameters:

- databaseconfiguration

Command output:
If the command is successful, a JSON-encoded object is returned representing the current database configuration.

Minimum privilege level required:
System Administrator

Curl example:

curl -i -u admin:admin http://<server-hostname>:6580/api/databaseconfiguration

PUT

Creates or updates the configuration used to connect to a database server.

Command syntax and parameters:

- databaseconfiguration

  The POST body contains a JSON-encoded object representing the new or updated database configuration.

Request body should contain a JSON object with attributes

type (required) - <"POSTGRESQL" | "SQLSERVER">

host (required) - The hostname or IP of the database server

port (required) - The port of the database server

name (required) - The name of the database

user (optional) - The username used to connect to the database

password (optional) - The password used to connect to the database

contact (optional) - An email address to receive database disconnection email alerts

additional (optional) - Database specific properties. Separate properties using semicolon (;) for MSSQL or ampersand (&) for PostgreSQL

intent (optional) - <"primary" | "standby"> The intended operation mode this CenterScape instance should use.

Command output:

If the command is successful, no output is returned from the command. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object update failed.

Minimum privilege level required:

System Administrator

Curl example:

```
curl -i -X POST -u admin:admin http://<server-
hostname>:6580/api/databaseconfiguration
```

POST

N/A

DELETE

N/A

5.9. Entity Type

COMMAND

<base-url>/entitytype

DESCRIPTION

The entitytype command is used to create, update or delete entity type objects.

GET

Retrieves one or more entity types.

Command syntax and parameters:
General parameters:

- format = <“JSON”|“CSV”> Set the response format.

- entitytype?children=true – Returns entity types which do not have a parent.

  children = <true> (optional) – Return all child entity types of the top-level entity types.

- entitytype/<object-type-guid>?inherited=<true|false>

  object-type-guid (required) The guid of the entity type to retrieve.

  Returns the specified entity type and all its children unless the inherited parameter is set to false.

  inherited = <true|false> (optional) – Returns the entity types the specified entity type inherits from.

Command Output:

The output of the command is one or more entity types encoded using the requested format. The default format is JSON.

Minimum privilege level required:

Asset Viewer

Curl example:
curl -i -u admin:admin http://<server-hostname>:6580/api/entitytype?children=true

POST

Creates a new entity type.

Command syntax and parameters:

- entitytype

The POST body should contain a JSON-encoded object representing the new entity type.

Example:

{

"guid" : "TRUCK",

"name" : "Truck",

"description" : "",

"deletable" : true,

"parent" : "$tAsset"

}

Command output:

If the command is successful, no output is returned from the command. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object update failed.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X POST -H 'content-type: application/json' -u admin:admin -d @<post-body-filename> http://<server-hostname>:6580/api/entitytype

Where <post-body-filename> is the name of the file containing the post body.

PUT

Updates an existing entity type.

Command syntax and parameters:

- entitytype/<entity-type-guid>

The POST body contains a JSON-encoded object representing the updated entity type.

Example:

```
{
        "guid" : "TRUCK",
        "name" : "Truck",
        "description" : "",
        "deletable" : true,
        "parent" : "$tAsset",
        "children" : [],"attributes" : [{"guid" : "COLOR",
                "name" : "Color",
                "deletable" : true,
                 “required" : true,
                "isStatic" : false,
                "category" : "Appearance",
                "sortPriority" : 500}],
        "class" : "entity_type"
}
```


Command output:

If the command is successful, no output is returned from the command. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object update failed.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X PUT -H 'content-type: application/json' -u admin:admin -d @<post-body-

filename> http://<server-hostname>:6580/api/entitytype

Where <post-body-filename> is the name of the file containing the post body.

DELETE

Deletes an existing entity type.

Command syntax and parameters:

- entitytype/<entity-type-guid>

  entity-type-guid (required) – The guid of the entity type to delete.

Command output:

If the command is successful, no output is returned from the command. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object update failed.

Minimum privilege level required:

System Administrator

Curl example:

curl -i -X DELETE -u admin:admin http://<server-hostname>:6580/api/entitytype/entity-type-guid

5.10. Filter

COMMAND

<base-url>/filter

DESCRIPTION

The *filter* command allows you to create a query by creating a number of conditions based on asset type, asset location, asset attribute values, and other asset states. The assets which match all the conditions specified in the filter are included in the result set returned to the command caller.

The most basic form of the filter command is querying assets by type. In fact, asset type is the only required condition that must be specified in the filter command, all other parameters are optional.

There are two methods the filter command can use. The first method allows you to construct a filter and then query which assets on the CenterScape server match the condition(s) specified in the filter. When the command is run the assets that match the conditions specified in the filter are returned in the result set.

The second method allows you to declare the conditions an asset must match in order to be returned in the result set just like the first method; however, by specifying a unique value for the *filterid*, you are instructing the CenterScape server to remember the filter definition and accumulate change notifications to the result set initially returned. You can then issue another command – *filterupdates* – to listen for these change notifications to the assets that were originally returned from the filter command.

The *filter* and *filterupdates* commands use session cookies to correlate the calling application across multiple requests. Normally, if the commands are called from a web page, the browser will manage the session cookie. When not using a browser, the session cookie can be specified by appending “;jsessionid=<session-cookie>” on the end of a URL but before any parameters.

You can think of the *filter* command as a means of defining a view into the set of assets in the CenterScape server. Using a *filterid* parameter, the server will accumulate change notifications to the view. You can think of the filterupdates command as the means of receiving updates for the currently defined filters.

GET

Query assets by type and location, or any arbitrary attribute value.

Command syntax and parameters:

- Filter

type = <entity-type-guid> (required) – The base type of the objects for which to receive change notifications.

location = <location-type-guid> (optional) – The entity type guid of the location object the entities must equal to in order to be included in the result set. For example, if the location is FLORIDA, only entities that have a location of FLORIDA would be included in the result set.

attribute = <attribute-guid> (optional) Specifies the guid of the attribute used in the filter condition.

operator = <“eq” | “ne” | “gt” | “lt” | “ge” | “le”, “undef”| “def”, “contains” | “containselement” | “startswith” | “in”> (optional) Specifies the operator used when comparing the value of the filter attribute.

value = <attribute comparison value> (optional) Specifies the value to compare in the filter condition.

Note: The attribute, operator, and value parameters must either all be specified or none of them specified. They allow for an additional condition to be specified for defining the attributes an entity must match in order to be included in a result set. For example, you may have an attribute named COLOR with allowable values of Red, Blue, Green. By specifying the attribute=“COLOR”, operation=“eq”, and value=“Red”, entities must match this condition to be included as part of the filter results.

attributeset = <attribute-guid1, attribute-guid2, ...> (optional) – Allows you to specify which attribute values the filter command should return for entities that match the filter criteria. If no attributeset is specified, only the entity guid is returned for each entity in the result set.

attributesort = <attribute-guid> (optional) – Allows you to specify the attribute whose value should be used when sorting the filters results.

start = <start-index> (optional) – The index in the result set from which to start returning results. For example, if the filter results in 500 entities matching and if the start value is 300, the result set would start at the 301st element in the set.

count = <maximum-result-count> (optional) – The maximum number of items that should be included in the result set.

Note: The start and count parameters are used to create a rolling view over a population of assets that match the filter condition(s). This is the mechanism the CenterScape web UI uses to view large asset populations.

sortorder= <“asc”|“desc”|“none”> (optional) – The order in which to sort the result set based on the attributesort.

showretired= <“yes”|“no”|“both”> (optional) – Indicates if retired entities should be included or excluded for the filter result.

filterid= <“uniqueid”> (optional) – This parameter is used to indicate you want the CenterScape server to accumulate change notifications for filter. All the change notifications are updates to the result set that was returned from this command. If this parameter is not specified, the filter is evaluated, its result sets are returned and the server does not retain any state for this filter.

Command output:
The output of the command is a JSON-encoded array containing the results of the filter query. There are additional objects encoded in the array which indicate how the response should be decoded. Each object returned contains an attribute called updtype which indicates information about the result set of the results themselves. The possible values of updtype are:

create – An asset entered the filter result set.
update – An asset that was in the result set has been updated or is part of the initial set returned.

delete – An asset has been removed from the result set as a result of the asset being deleted on the CenterScape server.

entered – An asset has entered the filter result set as a result of one or more the asset attributes being updated to match the filter condition(s).

exited – An asset has exited the filter result set as a result of the asset’s attribute(s) no longer matching the filter condition(s).

countupd – The number of assets which are returned from the command.

rowlist – If sorting was requested, the rowlist is the sorted order of the entity guids in the result set.

An example will illustrate how the command output is formatted. The following curl example is used to query all the assets of the root asset type $tAsset:

curl -i -u admin:admin http://<server-hostname>:6580/api/filter?type='$tAsset'&count=1

The results of the command are:


```
[
    {
    "class" : "fltupdevt", "timestamp" : 1225474644976, "updtype" :
    "countupd", "count" : 1
    },

    {
    "class" : "fltupdevt", "timestamp" : 1225474644976, "updtype" :
    "entered",
    "entity" : {
        "class" : "entity",
        "guid" : "CAR_fcf94f38",
        "retired" : false,
        "deletable" : true
        }
    },
    {
    "class" : "fltupdevt",
    "timestamp" : 1225474644976,
    "updtype" : "rowlist",
    "last-update" : true,
    "roworder" : [ "CAR_fcf94f38" ]

    }
]
```


Each object in the result set also includes the CenterScape timestamp of when the event was created.

The first object included in the result set is of an updtype of “countupd”. This object indicates how many asset updates are included in the response.

The next object that is returned is the single asset that matched the filter condition and in this example is bounded by the maximum count of 1. The asset itself is nested

in the object under the Entity property. For all the assets returned from this command, the updtype is “entered”.

The last update is a rowlist object which contains the roworder property. This is the new sort for the contents of the assets in the filter result set.

Minimum privilege level required:
The minimum privilege level required is dependent on the type of the objects being request by the filter’s type parameter. In general, Asset Viewer is required when specifying customer asset types. System types require System Administrator privilege.

Curl example:
curl -i -u admin:admin http://<server-hostname>:6580/api/filter?type=%24tAsset&filterid=fid_84133&attribute=%24aAssetTamper&operator=eq&value=TRUE&attributeset=%24aName&attributesort=%24aName&sortorder=asc&start=0&count=25

This example queries the assets whose tag tamper attribute are currently true. The name attribute is returned for each asset, the entities are sorted in ascending order by their name and the view starts at the first asset in the sorted list and returns a maximum of 25 assets.

POST

Same as GET. The parameters must all be URL-encoded in the body of the post.

PUT

N/A

DELETE

N/A

5.11. Filter Delete

COMMAND

<base-url>/filterdelete

DESCRIPTION

The filterdelete command is used to delete a filter that was previously created as interested in receiving change notifications. The server will release any resources allocated for accumulating change notifications for the filter.

GET

Delete a filter.

Command syntax and parameters:

- filterdelete?*filterid*=<filter id>

filterid = <filter id> (required) – The ID of the filter to delete. This value should match the *filterid* parameter value used when creating a filter.

Command output:
No output is returned from this command.

Minimum privilege level required:
Asset Viewer

Curl example:
curl -i -u admin:admin http://<server-hostname>:6580/api/filterdelete

POST

N/A

PUT

N/A

DELETE

N/A

5.12. Filter Update

COMMAND

<base-url>/filterupdate

DESCRIPTION

The filterupdate command allows to request asset change notifications for all filters created using the same session id and for filters that specified a value for the filterid parameter.

The filterupdate command is used to periodically drain the queue of accumulated change notifications for the saved filters. The caller is expected to invoke this command, wait for any updates that may have accumulated, process the change notifications received from the CenterScape server and repeat this process as long as the caller is interested in receiving change notifications.

Filter change notifications will be accumulated for saved filters up till the point where they are deleted using the filterdelete command or if they expire. A saved filter will expire if there has been no request to retrieve change notifications for a filter for 120 seconds.

GET

Request asset change notifications.

Command syntax and parameters:

- filterupdates?_timeout=<milliseconds>&_maxcnt=<maximum assets to return>

_timeout = <millisecond> (optional) – This parameter controls how long the server waits to accumulate filter change notifications before returning to the caller. If the server already has change notifications queued when *filterupdates* is called, those results are immediately returned to the caller. If there are no pending change notifications, the server waits up to the timeout period or until a change notification is queued before returning to the caller. The default timeout is 30 seconds (30000 milliseconds). _maxcnt = <count> (optional) – This parameter is used to bound the size of the change notifications returned in any single call to the filterupdates command. If this parameter is not specified, all queued change notifications are returned to the caller.

Command output:
The output of the command is a JSON-encoded array containing the results of the filter query. If there are no queued filter change notifications, an empty JSON array "[]"

is returned. There are additional objects encoded in the array which indicate how the response should be decoded. Each object returned contains an attribute called *updtype* which indicates information about the result set of the results themselves. The possible values of *updtype* are:

create – An asset entered the filter result set.
update – An asset that was in the result set has been updated or is part of the initial set returned.
delete – An asset has been removed from the result set as a result of the asset being deleted on the CenterScape server.
entered – An asset has entered the filter result set as a result of one or more the asset attributes being updated to match the filter condition(s).
exited – An asset has exited the filter result set as a result of the asset's attribute(s) no longer matching the filter condition(s).
countupd – The number of assets which are returned from the command.
rowlist – If sorting was requested, the rowlist is the sorted order of the entity guids in the result set.

In the following example, a new *CAR* asset has been added. One of the filters defined has requested to listen for all assets of type CAR. The filter was created with its *filterid* = "fid_63020". If you have multiple filters defined for which you are listening for change notifications, you should expect to have change notifications for the filters returned in the same response from *filterupdates*. This does not mean the notifications with in the same update will be interleaved. The results for each filter update will be sequential. You will process the change notifications for one filter keyed off of the *filterid*, and then the next keyed off of *filterid*, and so on until there are no remaining notifications.

The following is a response to *filterupdates* as a result of a new Car asset type being created.


```
[
        {
                "class" : "fltupdevt", "filterid" : "fid_63020",
                "timestamp" : 1225481827543, "updtype" : "countupd", "count" : 2
        },
        {
                "class" : "fltupdevt", "filterid" : "fid_63020",
                "timestamp" : 1225481827543, "updtype" : "create",
                "entity" : {
                        "class" : "entity",
                        "guid" : "CAR_76c931e4",
                        "retired" : false,
```

```
                        "deletable" : true,
                        "$aName" : "F-150",
                        "$aDescription" : "",
                        "type" : "CAR"
                        }
            },
            {
            "class" : "fltupdevt",
            "filterid" : "fid_63020",
            "timestamp" : 1225481827543,
            "updtype" : "rowlist",
            "last-update" : true,
            "roworder" : [ "CAR_76c931e4", "CAR_fcf94f38" ]
            }

]
```


Minimum privilege level required:

Asset Viewer. The filter command will control the privilege of the assets which can be requested by a user and in turn receive change notifications using this command.

Curl example:
curl -i -u admin:admin http://<server-hostname>:6580/api/filterupdates

POST

Same as GET. The parameters are URL-encoded in the body of the post.

PUT

N/A

DELETE

N/A

5.13. History

COMMAND

<base-url>/history

DESCRIPTION

This *history* command is used to report all value changes for all assets or a subset of assets for a specified time range. The population of assets reported on can be subset by asset type and optionally asset location. The time range specified for the API can be an absolute time range (for example, February 1, 2014 to February 28, 2014) or relative to the current time (for example, the last hour relative to the time the API is invoked).

The API can optionally be invoked to report the lead-in value for all assets being reported on. The lead-in value is described as the value of an attribute on or before the start time of the report. If the lead-in value is already known, it is recommended to not report the lead-in value due to the additional resource expense required to query these values.

GET

Retrieves the asset changes during the time range specified in the command for all or the specified list of attributes.

Command syntax and parameters:

- history?filtertype=tAsset&seconds=600&name=LastTenMinutes&prettyprint=false&format=CSV&leadin=true

filtertype (required) – The guid of the asset type specifying the assets to report on. If the intent is to report on all assets, the value of this parameter should be "$tAsset".

filterlocation (optional) – The guid of a location. This allows the caller to subset the population of assets being returned by the location specified.

name (optional) – The name of the report. Used to give a name to the report when the report output is JSON.

seconds (optional) – The number of seconds indicating the range of the report relative to the current time. For example, if you want to report history for the last ten minutes, the value of seconds is 600.

start (optional) – The start time of the report in milliseconds. (Epoch time)

stop (optional) – The stop time of the report in milliseconds. (Epoch time)

calendar (optional) – Allows the user to specify the start/stop time of the report using convenient calendar values. The values for calendar are:

thisday – The current day from midnight to the current time.

thisweek – The current week from Sunday midnight to the current time. *thismonth* – The current month from the first of the month to the current time. *yesterday* – The beginning of yesterday to the end of yesterday.
lastweek – The beginning of the start of last week to the end of last week.

lastmonth – The beginning of the start of last month to the end of last month.

prettyprint (optional) – Used to display the raw value of an attribute or the cooked value of an attribute relative to the servers timezone/locale. The value is either true or false.

format (optional) – Used to specify the output format. The value is either JSON or CSV. The default is JSON.

leadin (optional) – Used to specify if the lead-in value should be reported for attributes. If the lead-in value is known, the value of the attribute should be false due to the additional expense incurred to produce the lead-in value.

attribute (optional) – The guid of the attribute(s) to report. If the attribute is omitted, all attributes which had a value change during the report interval are returned.

Command output:
The output of this command are the results of the attribute changes presented in either a JSON or CSV.

Minimum privilege level required:
The minimum privilege level required is dependent on the type of the objects in the report output. Since the report output may contain assets of different types, the requester must have the privilege level necessary to view all the assets contained in the report, otherwise the request will fail.

Curl example:
To report the history changes for all assets for the last ten minutes and output the result using CSV:

curl -u admin:admin 'http://<server-hostname>:6580/API/
history?seconds=600&filtertype= %24tAsset&name=
LastTenMinutes&prettyprint=false&format= CSV&leadin=true'

To report the history changes for all assets from the beginning of May 1st, 2014 to the end of May 1st 2014:

curl -u admin:admin 'http://<server-hostname>:6580/API/ history?start=1398902400000&stop= 1398988799999&filtertype= %24tAsset&name= MayFirst2014&prettyprint=false&format= CSV&leadin=true'

POST

N/A

PUT

N/A

DELETE

N/A

5.14. Report Output

COMMAND

<base-url>/reportoutput

DESCRIPTION

The reportoutput command is used to generate report output from a previously created report.

GET

Retrieves (generates) a report for a specified report object.

Command syntax and parameters:

- reportoutput/*<report-guid>*.<extension>?start=<start index>&lim-it=<max results>

report-guid (required) – The guid of the existing report object.
extension <"json" | "csv" | "pdf" | "xml" | "png"> (optional) – Indicates the type of report output to generate. The default output format is JSON.

start = <start index> (optional) – The starting index in the result set for which to return results. The default value is 0.
limit = <max results> (optional) – The maximum number of items to include in the report output. The default is to include the entire report results in the report output.

Command output:
The output of this command are the results of the report presented in either a JSON, CSV, PDF, XML, or PNG image file

Minimum privilege level required:
The minimum privilege level required is dependent on the type of the objects in the report output. Since the report output may contain assets of different types, the requester must have the privilege level necessary to view all the assets contained in the report, otherwise the request will fail.

Curl example:
curl -i -u admin:admin 'http://<server-hostname>:6580/api/reportoutput/$tAssetReportJob_a44efddc.csv'

curl -i -u admin:admin 'http://<server-hostname>:6580/api/reportoutput/$tAssetReportJob_a44efddc.xml?start=10'

POST

N/A

PUT

N/A

DELETE

N/A

5.15. Report

COMMAND

<base-url>/report

DESCRIPTION

The *report* command is used to output an instant report for the specified entity and attribute.

GET

Retrieves (generates) a report for a specified report object.

Command syntax and parameters:

- report.<extension>?entity=<entity-guid>&attr=<attribute-guid>&msecsAgo=<milliseconds-ago>&name=<name>

extension <"json" | "csv" | "pdf" | "xml" | "png"> (optional) – Indicates the type of report output to generate. The default output format is JSON.

entity = <entity-guid> (required) – The guid of the entity to report on.

attr = <attribute-guid> (required) – The guid of the attribute to report on.

msecsAgo = <milliseconds-ago> (required) – The number of milliseconds before current time to start the report.

name = <name> (optional) – The name to display with the report output.

Command output:
The output of this command are the results of the report generated in either a JSON, CSV, PDF, XML, or PNG image file.

Minimum privilege level required:
The minimum privilege level required is the same as the privilege of the entity-guid used when creating the instant report.

Curl example:
curl -i -u admin:admin 'http://<server-hostname>:6580/api/report.xml?attr=$aAssetMotion&entity=CAR_fe986be7&msecsAgo=21600000&name=Instant%20Report%20Example'

POST

N/A

PUT

N/A

DELETE

N/A

5.16. Tag

COMMAND

<base-url>/tag

DESCRIPTION

The *tag* command is used to retrieve detailed diagnostic information for a specific RFID tag from the Zone Manager(s) connected to Asset Server.

GET

Retrieves detailed tag information for a specified tag.

Command syntax and parameters:

- tag/<tag-guid>

Minimum privilege level required:

tag-guid (required) – The guid of the tag for which to retrieve detailed information.

Command output:

The output of the command is a JSON-encoded object containing the tag data retrieved from all Zone Managers which are within range of the tag. The top most properties of the object are the Zone Manager guids which are currently observing the tag. An example output where only the local Zone Manager is observing the tag is:


```
{
    "LOCAL_RANGER" : {
        "RFCLOC00005495" : {
        "id" : "RFCLOC00005495",
        "attributes" : {
        "tagid" : "00005495",
        "taggroupid" : "$zTagGroup_mantis04A_9c88adf9",
        "tagtype" : "mantis04A",
        "irlocator" : "000",
```

```
                "motion" : "false",
                "tamper" : "false",
                "panic" : "false",
                 "lowbattery" : "false",
                 "locationzone" : "",
                "confidencebyrule" : {
                },
                "confidencebyrule_resolved" : {
                }
        },

        "taglinks" : [ {
        "tagid" : "RFCLOC00005495",
        "channelid" : "$zReaderM200_7277a2b_channel_A",
         "ssi" : -66,
        "channelid_resolved" : "Desk CHANNEL_A"

        }, {
                "tagid" : "RFCLOC00005495",
                "channelid" : "$zReaderM200_7277a2b_channel_B", "ssi" : -87,
                "channelid_resolved" : "Desk CHANNEL_B"

                } ]

        },

        "zonemanager" : "Local Zone Manager"

        }

}
```


Minimum privilege level required:

CenterScape

Curl example:

curl -i -u admin:admin http://<server-hostname>:6580/api/tag/RFCLOC00005495

POST

N/A

PUT

N/A

DELETE

N/A

5.17. Tag Import

COMMAND

<base-url>/tagimport

DESCRIPTION

The *tagimport* command is used to import RFID tag IDs.

GET

N/A

POST

The post is either a multi-part form upload or contains in the body a JSON-encoded object which contains the guids of the tags to create.

Command syntax and parameters:

- tagimport?returnRawData=<"*true*" / *"false"*>

- **returnRawData** - **Optional** query parameter. If true, the server will respond with a JSON object with the following properties
    - **created -**The number of tags that were created
    - **existsBound -** The number of tags that already existed in the system and are associated with an asset
    - **existsUnbound -** The number of tags that already existed in the system and are not associated with an asset
    - **failed** - The number of tags that failed to import
    - **failedMessages** - Up to 10 error messages for tags that failed to import

If the POST is a multi-part form upload, the format of the tag file is:
tag.tagID

LOCATE00003845,24FE2A9B
...

If the POST is not a multi-part form upload, the body of the POST is expected to be a JSON-encoded array with the following format:
{
"tagids" : ["tagid1", "tagid2", ...]
}

Command output:

The output of the command is a JSON-encoded status object indicating how many of the tags were imported out of the request. Example:

```
{
        "success" : true,
        "messages" : [
                {
                        "code" : "tag.upload.success",
                        "message" : "The tag import was successful: 0 tag(s) were
                imported and 1 tag(s) already existed."
                }
        ]
}
```

Example output when using **returnRawData** query parameter:

```
{
  “created”: 3,
  “existsBound”: 2,
  “existsUnbound”: 3,
  “failed”: 1,
  “failedMessages”: ["ABCDEF00000001: No Matching Tag Group found."]
}
```

Minimum privilege level required:

Asset Manager

Curl example:

curl -i -X POST -H 'content-type: application/json' -u admin:admin -d @<post-body-filename> http://<server-hostname>:6580/api/tagimport

Where <post-body-filename> is the name of the file containing the post body.

PUT

N/A

DELETE

N/A

5.18. Schemas

COMMAND

<base-url>/schemas

DESCRIPTION

The schemas command is used to retrieve the names of the predefined schemas bundled with the server.

GET

Retrieves a list of built-in asset schemas bundled with the CenterScape server.

Command syntax and parameters:

- schemas

Command output:
A JSON-encoded array of strings where each string is the name of a schema.

Minimum privilege level required:
System Administrator

Curl example:
curl -i -u admin:admin http://<server-hostname>:6580/api/schemas

POST

N/A

PUT

N/A

DELETE

N/A

5.19. Cluster Instances

COMMAND

<base-url>/instance

DESCRIPTION

The *instance* command can be used to view, update, or remove CenterScape instances that are registered with the cluster.

GET

Command syntax and parameters:

- instance
  - Retrieve all instances.
- instance/<instanceId>
  - Retrieve a single instance by <instanceId>. <instanceId> is the instance_id identifier in 'conf/server.uuid', a configuration file in CenterScape deployment.
- Instance/isPrimary
  - Returns HTTP status code of 200 if the instance is in primary mode. Returns HTTP status code of 400 otherwise.
- Instance/isStandby
  - Returns HTTP status code of 200 if the instance is in standby mode. Returns HTTP status code of 400 otherwise.

Command Output:

'instance' and 'instance/<instanceId>' command will output a JSON list or a single "Instance" object. An "Instance" object has the following properties

- **instanceId -** The unique identifier of this instance
- **Hostname** - The hostname of the host machine
- **intent** - <"primary" | "standby">
- **status** - <"primary" | "standby">
- **sincePrimary –** UTC timestamp of the system clock in the default time zone. The last time the instance had its 'intent' set to 'primary'
- **SinceConnected –** UTC timestamp of the system clock in the default time zone. The last time the instance started and connected to the cluster
- **heartbeat** - UTC timestamp of the system clock in the default time zone. The last time the instance checked in with the cluster
- **swVersion**- The CenterScape version of this instance

Minimum privilege level required:

'/instance' and '/instance/<instanceId>' require Read-Only Administrator, Application Administrator, or System Administrator. 'instance/isPrimary' and 'instance/isStandby' are for status only and require none.

POST

Command syntax and parameters:

- instance/<instanceId>

Post body contains a JSON object with attributes

- **intent** - <"primary" | "standby"> The intended operation mode of this instance.

This command can be used to change the intent of a particular instance. This endpoint cannot be used to update any of the instance attributes. All instance attributes are controlled by the CenterScape instance itself.

Command Output:

Returns a JSON encoded "Instance" with the updated intent. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the update failed.

Minimum privilege level required:

System Administrator

DELETE

Command syntax and parameters:

- instance/<instanceId>

Command Output:

None

Minimum privilege level required:

System Administrator

5.20. License Key Import

COMMAND

<base-url>/licensekeyimport

DESCRIPTION

The *licensekeyimport* command can be used to import license keys by uploading a file.

GET

N/A

POST

This command consumes multipart form data. The file must be in the form-field named 'file'. The file can either be a zip archive or a csv file. If the file is a zip archive, it must have a file in its root directory named 'licenses.csv'.

Command syntax and parameters:

- licensekeyimport
  - Accepts multipart form data with required field
    - **file** – The file to be uploaded

Command Output:

Returns a JSON object with attributes

- **attempted** - The number of license keys that CS attempted to import
- **created** - A list of the license keys that were successfully imported as new keys
- **existing** - A list of the license keys that were not imported because they are already in the system
- **expired** - A list of the license keys that were not imported because they have expired
- **errors** - The object keys will be the license keys. The object values will be an error message explaining why the key was not imported

Minimum privilege level required:

System Administrator

PUT

N/A

DELETE

N/A

5.21. Zone Manger Import

COMMAND

<base-url>/zonemanagerimport

DESCRIPTION

The *zonemgrimport* command is used to import a Zone Manager's configuration. This allows Zone Manager users to quickly configure a CenterScape server based on their existing Zone Manager configuration.

GET

N/A

POST

Imports a zone manager configuration.

Command syntax and parameters:

- zonemgrimport

The POST is a multi-part form upload containing a Zone Manager's configuration files. Not every file is required but if present, they must match the names of the saved Zone Manager configuration files. The files names are, in order, "groups.csv", "readers.csv", "locations.csv", and "rules.csv".

Command output:
If the command is successful, no output is returned from the command. If the command is unsuccessful, an errors object is returned indicating the reason or reasons why the object update failed.

Minimum privilege level required:
System Administrator

Curl example:

curl -u admin:admin -F "filecontent=@<groups.csv>" http://<server-hostname>:6580/api/zonemgrimport

PUT

N/A

DELETE

N/A

5.22. Expression Attribute Recalculation

COMMAND

<base-url>/recalc

DESCRIPTION

The *recalc* command is used to add an expression attribute to the recalculation queue. All assets that have the expression attribute get queued for recalculation of the specified expression attribute when this gets called. This enables syncing up the

expression attributes to the current dependent values in case the previous recalculation was interrupted and could not complete.

GET

N/A

POST

N/A

PUT

Triggers recalculation of the given recalculated asset attribute.

Command syntax and parameters:

- recalc/<object-guid>

object-guid (required) – The guid of the attribute class to trigger recalculation.

Enqueues recalculation of the specified asset attribute on all assets that have the attribute as referenced by object guid.

Command output:
If the command is successful, the API responds with no content in the body.

If the command fails due to insufficient privilege, it responds with a plain text response suggesting that the operation is not allowed.

If the requested object-guid could not be found, or is not an expression attribute, it responds with message in plain text body suggesting the requested expression attribute could not be found.

Minimum privilege level required:
System Administrator

Curl example:

curl -u admin:admin -X PUT 'http://<server-hostname>:6580/api/recalc/$aLocationVerified'

DELETE

N/A

5.23. File Validation

COMMAND

<base-url>/validatefile

DESCRIPTION

The *validatefile* commend is used to ensure the format of a given file meets the standards of CenterScape's use cases. This means that given a particular file extension, the shape of the document is considered intact and coherent by the CenterScape server. Currently only CSV files are supported.

GET

N/A

POST

The post is a multi-part form upload containing a file for validation

Command syntax and parameters:

- validatefile

The POST is a multi-part form upload of file type csv

Command output:
For a successful job:

```
{
 "success" : true,
 "jobid" : 1274020974,
 "messages" : [ ]
}
```

For an unsuccessful job the success value will be false and an error file will be provided containing the error details:

```
{
 "success" : false,
```

```
 "jobid" : 1288825162,
 "fileName" : "import-errors-20260325222209.csv",
 "errors" : [ ]
}
```


Minimum privilege level required:
Asset Editor

Curl example:

curl -u admin:admin -F "filecontent=@<data.csv>" http://<server-hostname>:6580/api/validatefile

PUT

N/A

DELETE

N/A

5.24. Attribute Mapping

COMMAND

<base-url>/purchaseorders/attributemappings

DESCRIPTION

The *attributemappings* commend is used to convert purchase order data into an entity-like format which can be imported into CenterScape. If a quantity determinant is mapped, the results will be enumerated based on that quantity.

GET

N/A

POST

The POST body should be a JSON object containing row data from a purchase order as well as a mapping of column headers to rows. There is also a capability to map rows

Command syntax and parameters:

- purchaseorders/attributemappings

attributeMap – a dictionary whose keys represent columns names and values represent the CenterScape attribute that the column will map to

nameMacro - (Optional) allows for a string expression that will be evaluated per entry present in *rowData* and stored as $aName. If a mapping exists in *attributeMap* that targets $aName this property will be ignored. Valid macros include:

- TIMESTAMP
- DATE
- DAY
- MONTH
- YEAR
- TIME
- HOUR
- MINUTE
- SECOND
- MILLISECOND
- TIMEZONE_OFFSET
- COLUMN.<Column_Name>

rowData – A collection of data representing rows to be processed. Object keys are column names and values are the corresponding entry in the source row. All column names present in *attributeMap* must have at least one valid key-value pair in *rowData*

Command output:
Assume an input as follows

```
{
        “attributeMap”: {
                “serial_number”: “$aName”
        “,
        “rowData”: [{
                “data”: {
                        “serial_number”: “848764e5862547ff81a4005fb93a3280”
                },
                "type”: “EQUIPMENT”
        },{
                 “data”: {
```

```
                    “serial_number”: “f5d71b210b2c468b94ac023c5983e77e”
        },
            “type": “EQUIPMENT”
        }]
}
```


The output would be as follows


```
[{
        “$aName”: “848764e5862547ff81a4005fb93a3280”,
        “class”: “entity”,
        “type”: “EQUIPMENT”
},{
        “$aName”: “848764e5862547ff81a4005fb93a3280”,
        “class”: “entity”,
        “type”: “EQUIPMENT”
}]
```


Note that if a numeric quantifier attribute $aQuantity is specified then the results for each row will be multiplied and have an increasing count appended to their $aName value:


```
{
        “attributeMap”: {
            “product_name”: “$aName”,
“qty”: “$aQuantity”
        },
        “rowData”: [{
            “data”: {
                “product_id”: “RXI-1105J”,
        “qty”: 2
},
            "type”: “EQUIPMENT”
        },{
             “data”: {
                “prodcut_id”: “LX-2V-EQ”,
        “qty”: 3
            },
            “type": “EQUIPMENT”
        }]
}

[{
```

“$aName": “RXI-1105J 1”,
“class”: “entity”,
“type”: “EQUIPMENT”
}, {
“$aName": “RXI-1105J 2”,
“class”: “entity”,
“type”: “EQUIPMENT”
}, {
“$aName": “LX-2V-EQ 1”,
“class”: “entity”,
“type”: “EQUIPMENT”
}, {
“$aName": “LX-2V-EQ 2”,
“class”: “entity”,
“type”: “EQUIPMENT”
}, {
“$aName": “LX-2V-EQ 3”,
“class”: “entity”,
“type”: “EQUIPMENT”
}]

Minimum privilege level required:
Asset Editor

Curl example:
curl -i -X POST -H 'content-type: application/json' -u admin:admin -d @<post-body-filename> http://<server-hostname>:6580/api/purchaseorders/attributemappings

PUT

N/A

DELETE

N/A

5.25. Reader Bulk Update Password

COMMAND

<base-url>/bulkreaderupdates

DESCRIPTION

The *bulkreaderupdates* command provides a set of APIs to create and manage tasks for updating passwords for multiple connected readers. This allows selecting both online as well as offline readers. As the task starts to execute, it waits a few times for offline readers to account for the possibility of temporary network failure. For a successful password update to happen, the readers should already have authenticated sessions.

GET

Retrieve a list of tasks that have been created in the past. This also gives the status of each task in the list and the status of password update for each reader in that task.

Command syntax and parameters

- bulkreaderupdates

Command output:
If successful, this returns JSON body representing a list of tasks with their corresponding details.


```
[{
 "class" : "entity",
 "guid" : "$tReaderCredentialsUpdateTask_8998faad82c2ed22",
 "retired" : false,
 "deletable" : true,
 "$aLastUpdateUser" : "admin",
 "$aBulkUpdateReaders" : "[ {\n  \"hostname\" : \"readerhost\",\n  \"updateStatus\" :
\"UPDATED\",\n  \"ip\" : \"192.168.0.11\",\n  \"guid\" :
\"$zReaderM250_9f7d6d2a6bb1b935\",\n  \"online\" : true,\n  \"errorMsg\" : \"\"\n} ]",
 "$aLastUpdateTime" : 1774276813615,
 "$aBulkUpdateReaderPassword" : "encrypted_password_set_during_bulk_update",
 "type" : "$tReaderCredentialsUpdateTask",
 "$aBulkUpdateCreator" : "ADMIN",
 "$aBulkUpdateCreationTimestamp" : 1774276808541,
 "$aBulkUpdateTaskStatus" : 3
}]
```


Minimum privilege level required:
Asset Viewer

Curl example:
curl -u admin:admin http://<server-hostname>:6580/api/bulkreaderupdates

GET

Retrieve details of a specific task created in the past. This works in a similar fashion as compared to the above API, just that this filters it down to a specific task and the status of the readers in that task.

Command syntax and parameters

- bulkreaderupdates/<task-guid>

<task-guid> (required) - Identifier of the bulk update task

Command output:
If not found, the API responds with error status. Otherwise, it returns the details of the task status as follows.


```
{
 "class" : "entity",
 "guid" : "$tReaderCredentialsUpdateTask_8998faad82c2ed22",
 "retired" : false,
 "deletable" : true,
 "$aLastUpdateUser" : "admin",
 "$aBulkUpdateReaders" : "[ {\n  \"hostname\" : \"readerhost\",\n  \"updateStatus\" :
\"UPDATED\",\n  \"ip\" : \"192.168.0.11\",\n  \"guid\" :
\"$zReaderM250_9f7d6d2a6bb1b935\",\n  \"online\" : true,\n  \"errorMsg\" : \"\"\n} ]",
 "$aLastUpdateTime" : 1774276813615,
 "$aBulkUpdateReaderPassword" : "encrypted_password_set_during_bulk_update",
 "type" : "$tReaderCredentialsUpdateTask",
 "$aBulkUpdateCreator" : "ADMIN",
 "$aBulkUpdateCreationTimestamp" : 1774276808541,
 "$aBulkUpdateTaskStatus" : 3
}
```


Minimum privilege level required:
Asset Viewer

Curl example:
curl -u admin:admin 'http://<server-hostname>:6580/api/bulkreaderupdates/$tReaderCredentialsUpdateTask_8998faa

d82c2ed22'

POST

Create task to update the readers password in a bulk

Command syntax and parameters:

- bulkreaderupdates

The POST accepts a JSON request that provides new password to update and the list of readers (identified by their guid) whose passwords need to be updated. All readers in the bulk update task will be configured using the same password. Request body would look like:

```
{
        "$aBulkUpdateReaderPassword": "password_to_update",
        "$aBulkUpdateReaders": [
                "$zReaderM250_91bbc736943a61de",
                "$zReaderM250_1320c1d9e91a0050",
        ]
}
```

Command output:
If the command is successful, it responds with JSON body with the details of the task created.
If the command is unsuccessful (e.g. when reader guid is not found or when no reader is specified), response text indicates the reason why the task creation failed.
If the user does not have sufficient permissions, it responds suggesting permission has been denied.

Minimum privilege level required:
System Administrator

Curl example:
curl -X POST -u admin:admin -H "Content-Type: application/json" -d '{"$aBulkUpdateReaderPassword": "password_to_update", "$aBulkUpdateReaders": ["$zReaderM250_91bbc736943a61de"]}' http://<server-hostname>:6580/api/bulkreaderupdates

PATCH

This allows for modification of the reader bulk update task. The only modification that is allowed though is to cancel the task if it is in progress.

Command syntax and parameters:

- bulkreaderupdates/<task-guid>

<task-guid> (required) - Identifier of the bulk update task

The API accepts a JSON request that provides the status that the task needs to be transitioned to. The task can be transitioned to CANCELLED state using the API.
Request body would look like below:

```
{
        "$aBulkUpdateTaskStatus": "CANCELLED"
}
```

Command output:
If the command is successful, it responds with no response body.
If the command is unsuccessful (e.g. when task has already transitioned to end state), it returns indicating the reason in the response body why the API failed.
If the user does not have sufficient permissions, it responds with error suggesting permission has been denied.

Minimum privilege level required:
Application Administrator

Curl example:

```
curl -X PATCH-u admin:admin  -H "Content-Type: application/json" -d '{"$aBulkUpdateTaskStatus": "CANCELLED"}' 'http://<server-hostname>:6580/api/bulkreaderupdates/$tReaderCredentialsUpdateTask_8998faad82c2ed22'
```

PUT

N/A

DELETE

N/A

6. Command Examples

This section demonstrates how to perform common queries and asset updates using the CenterScape command interface.

6.1. Exporting Assets by Filter – All Assets Attributes

DESCRIPTION

This example exports all customer assets in CSV format. Dates and times are exported using the ISO 8601 format. All asset attributes are exported since the attributeset parameter has a value of “*”.

GET

curl -i -u admin:admin 'http://<server-hostname>:6580/api/configexport?filter=true&format=CSV&type=%24tAsset&attributeset=**&sortorder=asc&showretired=no&dateformat=iso8601'

6.2. Exporting Assets by Filter – Specific Asset Attributes

DESCRIPTION

This example exports all customer assets in CSV format. Dates and times and exported using the ISO 8601 format. The assets attributes included are most of the tag specific attributes.

GET

curl -i -u admin:admin 'http://<server-hostname>:6580/api/configexport?filter=true&format=CSV&type=%24tAsset&attributeset=%24aName&attributeset=%24aAssetTag&attributeset=type&attributeset=%24aAssetTemperature&attributeset=%24aAssetPressure&attributeset=%24aAssetDoorOpen&attributeset=%24aAssetMotion&attributeset=%24aAssetTamper&attributeset=%24aAssetPanic&attributeset=%24aAssetLowBattery&sortorder=asc&showretired=no&dateformat=iso8601'

6.3. Creating Inheritance Maps

DESCRIPTION

This command creates a new Static Inheritance Map that can be associated with assets to allow them to inherit attribute values from multiple sources.

GET

curl -i -u admin:admin -H 'Content-type: application/json' -d '{'guid': 'INHERIT', 'name': 'Inherit', 'required': false, 'use': 'config-view','type': 'string', 'constraints': {'inheritancemap': [{'source': 'INHERIT', 'target': 'ASSIGNED_INHERIT', 'sortprio': 100, 'category': 'Basic Information'}]}}' -X PUT <server-hostname>:6580/api/attributeclass/INHERIT

6.4. Importing New Assets

DESCRIPTION

This example imports new assets from a CSV file.

GET

curl -i -u admin:admin -F 'filecontent=@asset.csv' 'http://<serverhostname>:6580/api/configimport'

The assets specified in the asset.csv will be uploaded and created.

Note: If the guid is not specified for the asset being created, a unique guid will be automatically generated. You can explicitly specify the guid if you so desire. If the guid for an asset already exists for an asset being imported, the attributes for the asset will be considered an update to the existing asset.

A sample CSV asset file would look like:
class,type,retired,deletable,$aName,COLOR,$aDescription
entity,CAR,false,true,Chevrolet Volt,Green,Plugin Hybrid

6.5. Updating Existing Assets

DESCRIPTION

This example is identical to creating assets. The sample asset file lists the guid for the assets that already exist. All the asset attribute values in the file are used to update the existing attributes.

GET

curl -i -u admin:admin -F 'filecontent=@asset.csv' 'http://<server-hostname>:6580/api/configimport'

A sample CSV asset file would look like:

class,guid,type,retired,deletable,$aName,COLOR,$aDescription entity,CAR_fe986be7,CAR,false,true,Chevrolet Volt,Red,Plugin Hybrid

6.6. Querying a Specific Asset By GUID

DESCRIPTION

This example queries a specific asset by its guid. The format of the output can be either CSV or JSON and is set using the *format* parameter. This example is requesting the output in CSV.

GET

curl -i -u admin:admin 'http://<server-hostname>:6580/api/entity/CAR_fe986be7?format=CSV'

6.7. Querying Assets by Location

DESCRIPTION

This example queries all assets with the location DEPOT. Like other examples which use a filter, you can select the type of assets you want the filter to match, the

specific attributes to include in the response, and the conditions the asset must match to be included in the response. The conditions in this example is the assets must be of *$tAsset* or a descendant of *$aAsset* and they must be in the location DEPOT or a descendant of DEPOT.

GET

curl -i -u admin:admin 'http://<server-hostname>:6580/api/filter?type=%24tAsset&location=DEPOT&attributeset=%24aName&attributeset=%24aAssetTag&attributeset=type&attributeset=%24aAssetTemperature&attributeset=%24aAssetPressure&attributeset=%24aAssetDoorOpen&attributeset=%24aAssetMotion&attributeset=%24aAssetTamper&attributeset=%24aAssetPanic&attributeset=%24aAssetLowBattery&sortorder=asc&showretired= no&dateformat=iso8601'

6.8. Creating a filter and Listening for Updates

DESCRIPTION

The following example creates a filter and using the filterid parameter instructs the CenterScape server to retain the filter information and start accumulating change notifications to the filter specified.

Note: The session cookie is specified using the jsessionid parameter. The filterupdate command uses the same value for the session id. The server will aggregate the change notifications for all filters created using the same session id and return the results.

GET

The following call creates the filter for all assets whose type inherits from the root Asset type and whose location is 'DEPOT'.

curl -i -u admin:admin 'http://<server-hostname>:6580/api/filter;jsessionid=SESSION_1?type=%24tAsset&location=DEPOT&attributeset=%24aName&attributeset=%24aAssetTag&attributeset=type&attributeset=%24aAssetTemperature&attributeset=%24aAssetPressure&attributeset=%24aAssetDoorOpen&

attributeset=%24aAssetMotion&attributeset=%24aAssetTamper&attributeset=%24aAssetPanic&attributeset=%24aAssetLowBattery&sortorder=asc&showretired=no&dateformat=iso8601?filterid=DEPOTLOCATION'

The following call will query for the updates to all the filters that were created using the sessionid = 'SESSION_1'. The command will return immediately if there are change notifications pending to any of the filters created using the same session id or wait the default time of 30 seconds before returning if no changes are pending.

curl -i -u admin:admin 'http://<server-hostname>:6580/api/filterupdates;jsessionid=SESSION_1'

6.9. Query All Unresolved Alerts

DESCRIPTION

The following example creates a filter to query for all alerts which are not resolved or closed.

GET

curl -i -u admin:admin 'http://<server-hostname>:6580/api/filter;jsessionid='ALERT_SESSION'?type=$tAssetAlert&attributeset=*&sortorder=asc&attribute=$aAlertState&operator=lt&value=3&filterid=ALERTS'

6.10. Create an Instant Report

DESCRIPTION

The following example runs an instant report on motion for a specific entity and outputs the results in XML format.

GET

curl -i -u admin:admin 'http://<server-

hostname>:6580/api/report.xml?attr=$aAssetMotion&entity=CAR_fe986be7&msecsAgo=21600000&name=Instant%20 Report%20Example'

6.11. Create a Report

DESCRIPTION

The following example creates a new Report entity.

GET

curl -i -X PUT -H 'content-type: application/json' -u admin:admin -d @update.json 'http://<server-hostname>:6580/api/entity'

This update creates a new Reader Noise Report for the last day displaying $aName, $zReaderNoiseA, and $zReaderNoiseB attributes for all readers on the system, assuming the reader has been in the NoiseDetected state during the last day using the JSON encoded object in the file update.json. The contents of the file are:

```
{

    "type" : "$tReaderNoiseReport"

    , "$aName" : "Reader Noise Example"

    , "$aReportEmailAddress" : ""

    , "$aReportEmailAttachment" : ""

    , "$aReportCreateOutputIfData" : true

    , "$aReportTime" : "$tReportTimeRangeRelative"
```

```
    , "$aReportTimeRangeRelative" : 3

    , "$aReportSchedule" : null

    , "$aReportScheduleEnable" : false

    , "$aReportAttribute" : ""

    , "$aReportAttributeValueOperator" : ""

    , "$aReportFilterAttribute" : ""
    , "$aReportFilterAttributeValueOperator" : ""

    , "$aReportPostconditionAttribute" : "$zReaderState"

    , "$aReportPostconditionAttributeValueOperator" : "eq"

    , "$aReportPostconditionAttributeValue" : 10

    , "$aReportChangeAttributes" : null

    , "$aReportAttributes" : ["$aName","$zReaderNoiseA","$zReaderNoiseB"]
}
```


7. Programming Tools

Java Jackson JSON parser:

https://github.com/FasterXML/jackson

CURL command tool and library

http://curl.haxx.se