



Governance API

Lifecycle Manager

Governance API

Version 7.0

July, 2015

Copyright

Copyright © 2015 Akana, Inc. All rights reserved.

Trademarks

All product and company names herein may be trademarks of their registered owners.

Akana, SOA Software, Community Manager, API Gateway, Lifecycle Manager, OAuth Server, Policy Manager, and Cloud Integration Gateway are trademarks of Akana, Inc.

Akana, Inc. (formerly SOA Software, Inc.)

Akana, Inc.

12100 Wilshire Blvd, Suite 1800

Los Angeles, CA 90025

(866) SOA-9876

www.akana.com

info@akana.com

Disclaimer

The information provided in this document is provided “AS IS” WITHOUT ANY WARRANTIES OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. Akana may make changes to this document at any time without notice. All comparisons, functionalities and measures as related to similar products and services offered by other vendors are based on Akana’s internal assessment and/or publicly available information of Akana and other vendor product features, unless otherwise specifically stated. Reliance by you on these assessments / comparative assessments is to be made solely on your own discretion and at your own risk. The content of this document may be out of date, and Akana makes no commitment to update this content. This document may refer to products, programs or services that are not available in your country. Consult your local Akana business contact for information regarding the products, programs and services that may be available to you. Applicable law may not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Contents

Chapter 1 Overview.....	6
Chapter 2 Asset Operations:	7
Get Initial Asset	7
HTTP Operation:.....	7
Path	7
Parameters:.....	7
Response	8
Create Asset	9
HTTP Operation:.....	9
Path	9
Parameters:.....	9
Request Body	9
Response	9
Update Asset	10
HTTP Operation:.....	10
Path	10
Parameters:.....	10
Request Body	10
Response	10
Get Asset	12
HTTP Operation:.....	12
Path	12
Parameters:.....	12
Response	12
Delete Asset	13
HTTP Operation:.....	13
Path	13
Parameters:.....	13
Response	13
Lock Asset.....	14
HTTP Operation:.....	14
Path	14
Parameters:.....	14
Response	14
Unlock Asset.....	15
HTTP Operation:.....	15
Path	15
Parameters:.....	15
Response	15
Get Assets	16

HTTP Operation:.....	16
Path	16
Parameters:.....	16
Response	16
Create Asset File.....	17
HTTP Operation:.....	17
Path	17
Parameters:.....	17
Request Body	17
Response	17
Update Asset File	18
HTTP Operation:.....	18
Path	18
Parameters:.....	18
Request Body	18
Response	18
Get Asset File.....	19
HTTP Operation:.....	19
Path	19
Parameters:.....	19
Response	19
Delete Artifact.....	20
HTTP Operation:.....	20
Path	20
Parameters:.....	20
Response	20
Chapter 3 Request Operations:.....	21
Submit Request	21
HTTP Operation:.....	21
Path	21
Parameters:.....	21
Request Body	21
Response	21
Update Request	22
HTTP Operation:.....	22
Path	22
Parameters:.....	22
Request Body	22
Response	22
Get Request.....	23
HTTP Operation:.....	23
Path	23

Parameters:.....	23
Response	23
Delete Request.....	24
HTTP Operation:.....	24
Path	24
Parameters:.....	24
Response	24
Get Requests	25
HTTP Operation:.....	25
Path	25
Parameters:.....	25
Response	25
Create Request File	26
HTTP Operation:.....	26
Path	26
Parameters:.....	26
Request Body	26
Response	26
Update Request File	27
HTTP Operation:.....	27
Path	27
Parameters:.....	27
Request Body	27
Response	27
Get Asset File.....	28
HTTP Operation:.....	28
Path	28
Parameters:.....	28
Response	28
Delete Artifact	29
HTTP Operation:.....	29
Path	29
Parameters:.....	29
Response	29

Chapter 1 | Overview

LM governance APIs are separated into operations on “assets” and “requests”. Assets are the entities being governed and support custom lifecycles. Requests represent a user’s “request” to promote the asset to its next lifecycle state. Requests are submitted then approved or rejected and may follow their own micro-lifecycle on the way to final approval of an asset transition.

The governance API attempts to simplify the client’s interaction with LM relative to LM’s own native clients. Some aspects of this simplification are:

- **Asset Structure**
Assets are represented in HTTP form style as a set of fields. These fields may be of simple types such as string, Boolean or decimal or may be of type File. The RAS-based classifier/artifact/relationship structure is hidden from the API client. Operations retrieving individual assets or requests will return the field data of the request entity along with field metadata in Alpaca format consisting of a JSON schema structure and an additional JSON “options” structure.
When passing in an asset or request, clients need only provide the entity data.
- **Locking**
The governance API uses an optimistic implicit locking model which assumes that not common that another user is concurrently updating assets maintained through the API. However, the API does expose explicit locking methods for a client that wants to manage locking.
- **Role Management**
The governance API assumes that the management of user’s and associated roles is the responsibility of the client application. While some API operations require user ids and role names, LM does not retain knowledge of the assignment of roles to users.
- **Notification**
In the initial release of the APIs there is no formal callback subscription model. Client applications may poll the “get” operations or install custom listeners within the LM configuration to manage callbacks.
- **By-value Artifacts**
Client applications may allow LM to implicitly manage File fields as “by-value” artifacts through use of the multi-part API operations or may choose to explicitly manage artifact using artifact operations. In either case the artifact-id is assumed to be the File field name. This implies that file fields must be single-valued.

Chapter 2 | Asset Operations:

The base URI for all APIs is <http://<host:port>/lm/rest/governance/<libraryname>>.

Get Initial Asset

This method is used to get the initial format information as well as any default data for the specified asset-type. Typically an application would call this method to produce the initial form to present to a user. While the asset in the response will have a unique id, the asset has not yet been created persistently within LM.

HTTP Operation:

GET

Path

/assets/new

Parameters:

- **asset-type (string)**
Type of the asset to create. Corresponds to an asset-type configured within LM.
- **user-id (string)**
The user requesting the asset. User id is optional and used only to allow customizations e.g. validators within LM access to the user.
- **role (string[])**
Roles the user possesses. These may optionally be used by the LM configuration for filtering the fields presented on the asset.
- **set-field (string[])**
Used to prime fields within the asset for the sake of tailoring the initial asset representation. For example, these field values may activate conditional fields within the LM configuration allowing different initial asset formats. Values are assumed to be in the form "<asset-field-name>:<asset-field-value>" and must specify valid asset fields for the asset-type. For example, specifying a parameter of "set-field=subtype:Accounting" will prime the asset field "subtype" with the value "Accounting".

Response

JSON

Initial asset data with asset-id, defaulted fields and field metadata.

Create Asset

Create a new asset within LM. By default the submission process for the initial state is triggered. If the submission process requires request input from the user the request data and format metadata prepared for the submitting user will be returned. If user input is not required on the request, only the request-id will be returned.

HTTP Operation:

POST

Path

/assets

Parameters:

- user-id (string)
The user submitting the asset. User id will be used to implicitly lock the asset and track asset change history. This parameter is optional and is defaulted to the LM application user.
- submit (boolean)
Indicates whether the asset should be submitted for the initial approval process. Default is "true".

Request Body

- JSON
Asset data in JSON format. File fields must be explicitly handled by client through artifact management API operations.
- Multipart/form-data
Asset and accompanying Files in multipart form style. Files will be implicitly created as artifacts

Response

JSON

If submit is "true" and submitter request input is required, the request id along with the request data and format metadata prepared for the submitter will be returned in a single JSON structure.

If submit is "true" and submitter request input is not required, the returned JSON structure will contain only the request-id.

If submit is "false", no data is returned.

Update Asset

Update an existing asset. By default the submission process for next approval process in the asset's lifecycle is triggered. If the submission process requires request input from the user the request data and format metadata prepared for the submitting user will be returned. If user input is not required on the request, only the request-id will be returned.

HTTP Operation:

POST

Path

/assets/{asset-id}

Parameters:

- user-id (string)
The user submitting the asset. User id will be used to implicitly lock the asset and track asset change history. This parameter is optional and is defaulted to the LM application user.
- submit (boolean)
Indicates whether the asset should be submitted for the next approval process. Default is "true".
- overwrite (Boolean)
Indicates that any changes to the asset since the asset was fetched by the client are overwritten. If "false" a locking error will be thrown if the asset is out-of-sync and asset will not be updated. The default value is "true".

Request Body

- JSON
Asset data in JSON format. File fields must be explicitly handled by client through artifact management API operations.
- Multipart/form-data
Asset and accompanying Files in multipart form style. Updated File fields (those with a value set) will be implicitly be created as artifacts or replace existing artifacts

Response

JSON

If submit is “true” and submitter request input is required, the request id along with the request data and format metadata prepared for the submitter will be returned in a single JSON structure.

If submit is “true” and submitter request input is not required, the returned JSON structure will contain only the request-id.

If submit is “false”, no data is returned.

Get Asset

Retrieve the current asset data and format metadata for an existing asset. By default this will be the data of the latest “in-progress” version of the asset in LM, but it is also possible to specify that the latest “approved” version of the asset be returned.

HTTP Operation:

GET

Path

/assets/{asset-id}

Parameters:

- role (string[])
Roles the user possesses. These may optionally be used by the LM configuration for filtering the fields presented on the asset.
- approved-version (Boolean)
A value of “true” indicates that the latest approved version of the asset in LM be returned (the “published” asset in LM terms). A value of “false” indicates that the latest “in-progress” version of the asset in LM be returned even if the approval of that version is not completed. The default value is “false”.

Response

JSON

Asset data and field metadata.

Delete Asset

Delete an existing asset.

HTTP Operation:

DELETE

Path

/assets/{asset-id}

Parameters:

- user-id (string)
The user deleting the asset. User id will be used to implicitly lock the asset prior to deletion and track asset change history

Response

NONE

Lock Asset

Explicitly lock an existing asset. If already locked by the specified user the operation has no effect. If the asset is locked by a different user the operation will fail.

HTTP Operation:

POST

Path

/assets/locks/{asset-id}

Parameters:

- user-id (string)
The user locking the asset. This parameter is optional and is defaulted to the LM application user.

Response

NONE

Unlock Asset

Explicitly unlock an existing asset. If the asset is already locked by the specified user the operation has no effect. If the asset is locked by a different user the operation will fail.

HTTP Operation:

DELETE

Path

/assets/locks/{asset-id}

Parameters:

- user-id (string)
The user unlocking the asset. This must match the user-id the asset is currently locked under. This parameter is optional and is defaulted to the LM application user.

Response

NONE

Get Assets

Query existing assets for those meeting the specified criteria. Data for specified fields is returned for each matching asset. The query may be specified to consider either in-progress or approved versions of assets based on “approved-version” parameter.

HTTP Operation:

GET

Path

/assets

Parameters:

- approved-version (Boolean)
A value of “true” indicates that only the latest approved versions of the assets in LM be queried. A value of “false” indicates that the latest “in-progress” versions of the assets in LM are queried. The default value is “false”.
- page (integer)
The number of the page of asset results to retrieve, starting with 1. The default value is “1”.
- page-size (integer)
The number of result assets to return per page. The default value is 500
- include-field (string[])
Names of fields to be included for each asset in the results
- order-by-fields
Ordered list of field names to use in ordering results. Specified fields should be string or decimal type. Field names are separated using “|”. For example “order-by-fields=field1|field2|field3”
- filter-field (string[])
A list of required values for designated asset fields names may be provided to filter the results of the query. Values are assumed to be in the form “<asset-field-name>:<asset-field-value>” and must specify valid asset fields for the asset-type. For example, specifying a parameter of “filter-field=color:blue” indicates that only assets with a “color” field value of “blue” should be selected. Multiple values may be specified for a single field by specifying them in multiple parameters with the same field name. In this case the value will be logically OR’ed together. For example, specifying “&filter-field=color:blue& filter-field=color:green” will match assets with either “blue” or “green” as a value for the “color” field.

Response

JSON

A JSON structure with an array of asset data elements containing the fields specified in the include-field parameters.

Create Asset File

Create a “by value” File in LM for the specified asset field. Used by clients explicitly managing asset File fields.

HTTP Operation:

POST

Path

/assets/{asset-id}/files/{field}

Parameters:

NONE

Request Body

Multipart/form-data
File contents

Response

NONE

Update Asset File

Update a “by value” File in LM for the specified asset field. Used by clients explicitly managing asset File fields.

HTTP Operation:

PUT

Path

/assets/{asset-id}/files/{field}

Parameters:

NONE

Request Body

Multipart/form-data
File contents

Response

NONE

Get Asset File

Retrieve a “by value” File in LM for the specified asset field. Used by clients explicitly managing asset File fields.

HTTP Operation:

GET

Path

/assets/{asset-id}/files/{field}

Parameters:

NONE

Response

Application/octet-stream

File contents.

Delete Artifact

Delete a “by value” File in LM for the specified asset field. Used by clients explicitly managing asset File fields.

HTTP Operation:

DELETE

Path

/assets/{asset-id}/files/{field}

Parameters:

NONE

Response

NONE

Chapter 3 | Request Operations:

Submit Request

Clients should use this operation to submit a request completed by the submitting user of an asset.

HTTP Operation:

POST

Path

/requests

Parameters:

- user-id (string)
The submitting user. User id will be used in the request history. This parameter is optional and is defaulted to the LM application user.

Request Body

- JSON
Request data in JSON format. File fields must be explicitly handled by client through File management API operations.
- Multipart/form-data
Request and accompanying Files in multipart form style. Files will be implicitly created as File properties

Response

NONE

Update Request

Update, approve or reject an existing request. The type of action performed is determined by the “action” parameter.

HTTP Operation:

POST

Path

/requests/{request-id}

Parameters:

- user-id (string)
The user performing the action. User id will be used in the request history. This parameter is optional and is defaulted to the LM application user.
- action (string)
Indicates whether the user is approving, rejecting or just updating the request. Acceptable values are “approve”, “reject” or “update”. The default value is “update”.
- approver-role (string)
Role for which the specified user is approving or rejecting the request. This parameter must be provided for “approve” or “reject” actions and must identify a currently pending role for the specified request.

Request Body

- JSON
Request data in JSON format. File fields must be explicitly handled by client through File management API operations.
- Multipart/form-data
Request and accompanying Files in multipart form style. Files will be implicitly created as File properties

Response

NONE

Get Request

Retrieve the current data and format metadata for an existing request.

HTTP Operation:

GET

Path

/requests/{request-id}

Parameters:

- role (string[])
Roles the user possesses. These may optionally be used by the LM configuration for filtering the fields presented on the request.
- approver-role (string)
In the case where the request is being retrieved for approval or rejection, this specifies the role the request should be prepared for. This parameter must identify a currently pending role for the specified request. In the case where the request is being retrieved for display or update this parameter may be omitted.

Response

JSON

Request data and field metadata.

Delete Request

Delete an existing request.

HTTP Operation:

DELETE

Path

/requests/{request-id}

Parameters:

NONE

Response

NONE

Get Requests

Query existing requests for those meeting the specified criteria. Data for specified fields is returned for each matching request.

HTTP Operation:

GET

Path

/requests

Parameters:

- requester-id (string)
User Id of the submitting user (optional).
- asset-id (string)
Id of the asset the request is associated with (optional).
- pending-role (string)
Name of a role the request is currently pending approval from (optional).
- page (integer)
The number of the page of request results to retrieve, starting with 1. The default value is "1".
- page-size (integer)
The number of result requests to return per page. The default value is 500
- include-field (string[])
Names of fields to be included for each request in the results
- order-by-fields
Ordered list of field names to use in ordering results. Specified fields should be string or decimal type. Field names are separated using "|". For example "order-by-fields=field1|field2|field3"
- filter-field (string[])
A list of required values for designated request fields names may be provided to filter the results of the query. Values are assumed to be in the form "<asset-field-name>:<asset-field-value>" and must specify valid asset fields for the request. For example, specifying a parameter of "filter-field=color:blue" indicates that only requests with a "color" field value of "blue" should be selected. Multiple values may be specified for a single field by specifying them in multiple parameters with the same field name. In this case the value will be logically OR'ed together. For example, specifying "&filter-field=color:blue& filter-field=color:green" will match requests with either "blue" or "green" as a value for the "color" field.

Response

JSON

A JSON structure with an array of request data elements containing the fields specified in the include-field parameters.

Create Request File

Create a “by value” File in LM for the specified request field. Used by clients explicitly managing request File fields.

HTTP Operation:

POST

Path

/requests/{request-id}/files/{field}

Parameters:

NONE

Request Body

Multipart/form-data
File contents

Response

NONE

Update Request File

Update a “by value” File in LM for the specified request field. Used by clients explicitly managing request File fields.

HTTP Operation:

PUT

Path

/requests/{request-id}/files/{field}

Parameters:

NONE

Request Body

Multipart/form-data
File contents

Response

NONE

Get Asset File

Retrieve a “by value” File in LM for the specified request field. Used by clients explicitly managing request File fields.

HTTP Operation:

GET

Path

/requests/{request-id}/files/{field}

Parameters:

NONE

Response

Application/octet-stream

File contents.

Delete Artifact

Delete a “by value” File in LM for the specified request field. Used by clients explicitly managing request File fields.

HTTP Operation:

DELETE

Path

/assets/{asset-id}/files/{field}

Parameters:

NONE

Response

NONE