CA Live API Creator DevOps Guide

Understanding CA Live API Creator Meta Model v5.0

Tyler M. Band
Principal Engineer
CA Technologies
Version 5.0 August 2018

Table of Contents

Overview

CA Live API Creator is a complete REST based service for authoring and publishing end-user API's. The reactive logic, security, connectivity to various SQL and Non-SQL endpoints and custom resources creates a remarkably flexible developer tool. What people may not know is that the entire platform itself is built using its own internal REST API endpoints, SQL Datasource, logic (aka rules), and security. The introduction of the 5.0 'serverless' design replaces the external SQL database to store API meta data with an in-memory Derby database. The meta data is now written to the file system in JSON, JavaScript, SQL, and HTML files.

- The 'sa' admin account logon will reveal the internal view of the system schema (aka meta model) named "CA Live API Creator Admin project".
- The admin schema consists of ~75 SQL tables stored in an in-memory Apache Derby database.
- The admin account uses Resources, Rules, Functions, and Security to control access to defined API components which in turn are used by the Live API Creator user interface.
- All the CA Live API Creator admin REST calls can be seen using the Live API Creator or tools like CURL, NPM liveapicreator-admin-cli, or POSTMAN using the admin project endpoint (http://{server}[:port]/*rest/abl/admin/v2*).

This book will explore the various key concepts used by the admin account to create and store meta objects and provide command line examples; e.g. what is the function or REST endpoint where is it stored in the admin database, how is it used, and a devops look at managing the service using Live API Creator admin command line (cli) utility (aka **lacadmin**).

While all the API information is stored in the in-memory SQL tables, there are special REST endpoints and custom Resources that will return your information in JSON format. This information is now stored locally on the filesystem which can be checked into a version control system, manually merged and edited, and migrated to new Live API Creator systems.

NodeJS

NodeJS is a JavaScript library that was used to create several command line tools (referenced here as lacadmin, lac, and apicreatorsdk). Install the admin command line utility using the NPM (Node Package Manager) package installer:

```
$npm install liveapicreator-admin-cli –g

$npm install liveapicreator-cli –g

$npm install APICreatorSDK -g
```

Many of the examples show both the internal SQL meta model and the lacadmin command line service. Each service provides some functionality to list, export, or import JSON content for the specific file or service. (Note: MAC OSX users may need to use $sudo npm install [package]).

Compiled Node Commands

The devops community has requested a compiled version of the NodeJS command line tools. These tools can be found here (https://github.com/EspressoLogicCafe/MigrationService). This directory contains 3 sub-directories which contain the NodeJS compiled versions of 'lacadmin' and 'lac' command line tools. Copy these 2 compiled files to your path for global access.

```
git clone https://github.com/EspressoLogicCafe/MigrationService.git
```

Live API Creator Admin Command Line

The NodeJS library follows a convention that makes these commands self-documenting. Use the double dash (--help) to see the current available commands (lacadmin –help) or to see a specific command use the syntax (lacadmin [command] –help).

Login

The lacadmin command line utility will store your logon credentials for one or more connections. (The files are stored in a folder under your home directory named ($HOME/.liveapicreator). The optional –a (alias) flag will name your connection for future use. This allows multiple connections to different severs to be managed concurrently. These examples use the syntax for MAC OSX:

```
$lacadmin login -u admin -p Password1 http://localhost:8080 -a jetty
Logging in...
This server licensed to: CA Technologies license_type: TRIAL
Login successful, API key will expire on: 2018-12-30T01:42:47.216Z
```

```
$lacadmin use jetty
You are now using server http://localhost:8080/rest/abl/admin/v2 as user admin

$lacadmin status
Defined aliases:
┌─────┬──────────────────────────────────────┬─────┐
│Alias│Server                                │User │
├─────┼──────────────────────────────────────┼─────┤
│jetty│http://localhost:8080/rest/abl/admin/v2│admin│
└─────┴──────────────────────────────────────┴─────┘
You are currently logged in to admin server: http://localhost:8080/rest/abl/admin/v2 as user admin
There is no current project.

$lacadmin api use --url_name demo
Current project is now: Demo (2000)
```

The login command requires a username, password and LAC server URL base name. The optional alias (-a) flag allows multiple connections to different servers. The next command '*lacadmin use*' will switch connection context to the selected alias (you can have multiple connections active). Finally, the '*lacadmin status*' command gives a good visual report of the connections, servers, and alias settings. If you select a specific API to use (i.e. lacadmin project use [--url_name (API url fragment)]), then all future commands which may require a *project_ident* will use the current select project (use **`lacadmin api list`**).

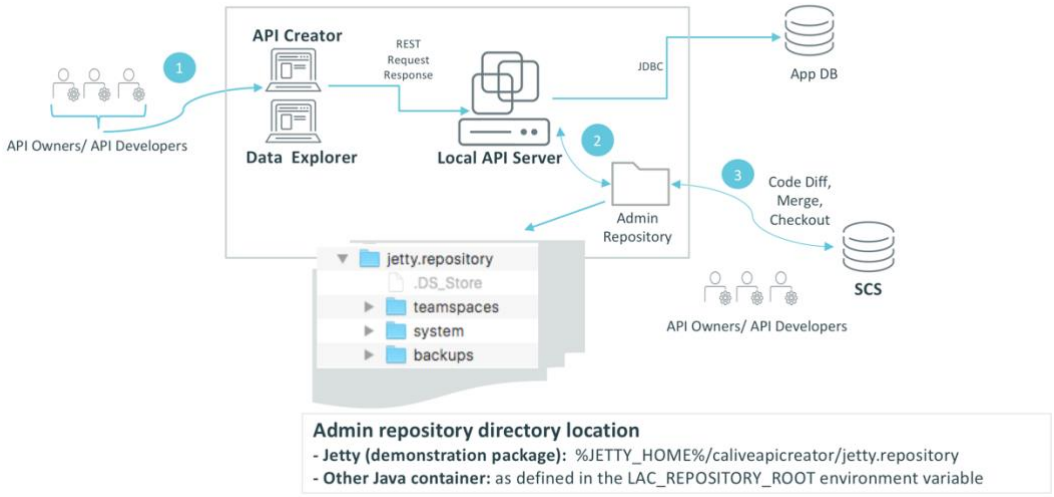
REST Call: http://server[:port]/[war_root_name]/rest/abl/admin/v2/@authentication

In-memory Derby Admin Database

The 5.0 CA Live API Creator (LAC) stores all definitions in an Apache Derby in-memory SQL database. In fact, the system will use 2 different schema entries (LACAdmin and LACApiKey). On first start, the system will look for a directory on the local machine:

WAR: {USER_HOME}/CALiveAPICreator.repository

Jetty package: {JETTY_HOME}/CALiveAPICreator/jetty.repository.



You can control the location of your root repository by setting the environment variable below:

```
LAC_REPOSITORY_ROOT=/tmp/CALiveAPICreator.repository
```

The system can backup changes to the filesystem.

- **LAC_REPOSITORY_BACKUP=true**
- **LAC_REPOSITORY_BACKUP_DAYS_TO_KEEP=10**

The root repository will contain 3 folders /backup, /system, and /teamspaces.

File Formats

The lacadmin utility was written to export/import 'raw' or database values. The file system repository stores the data files in an enhance JSON file format. You can use lacadmin scripts to export/import between versions or server endpoints. The new repository file system makes editing, copying, and deleting components much easier. If you need to export a specific 'section' use lacadmin, see lacadmin api export –section [section_name]. The style and names are similar but not an exact. The file system organizes the directory based on the relationship hierarchy.

Database Format	JSON Enhance Format
{	{

<table>
<tr>
<td>

<pre>
    "ts": "2018-07-05T21:44: ",
    "conn_type": null,
    "prefix": "demo",
    "name": "demo",
    "title": "Demo",
    "comments": null,
    "datasource_name": null,
    "url":
"jdbc:derby:directory:/Demo",
    "catalog_name": null,
    "schema_name": "DEMO",
    "table_includes": null,
    "table_excludes": null,
    "proc_includes": null,
    "proc_excludes": null,
    "user_name": "DEMO",
    "password":
"2:Sov+pD78k9RnLuVhW1==",
    "properties": null,
    "salt": "jZLKrayshHUs0F3INR",
    "schema_editable": false,
    "read_only": false,
    "active": true,
    "provider_ident": null,
    "provider_param_map": null,
    "support_boolean": false,
    "role_strategy": "simplicity",
    "log_errors": false,
    "status": null,
    "max_connections": 20,
    "project_ident": null,
    "dbasetype_ident": 17

  }
</pre>

</td>
<td>

<pre>
        "name": "demo",
        "prefix": "demo",
        "title": "Demo",
        "databaseType": "DERBY",
        "comments": null,
        "isActive": true,
        "isLogErrors": false,
        "isSchemaEditable": false,
        "isSupportBoolean": false,
        "isReadOnly": false,
        "useJNDIDataSource": false,
        "url":
"jdbc:derby:directory:/Demo",
        "catalog": null,
        "schema": "DEMO",
        "username": "DEMO",
        "maximumConnections": 20,
        "tableExcludes": null,
        "tableIncludes": null,
        "procedureExcludes": null,
        "procedureIncludes": null,
        "nonPersistentAttributes":
{},
        "tableInfos": {},
        "viewInfos": {},
        "procedureInfos": {}

   }
</pre>

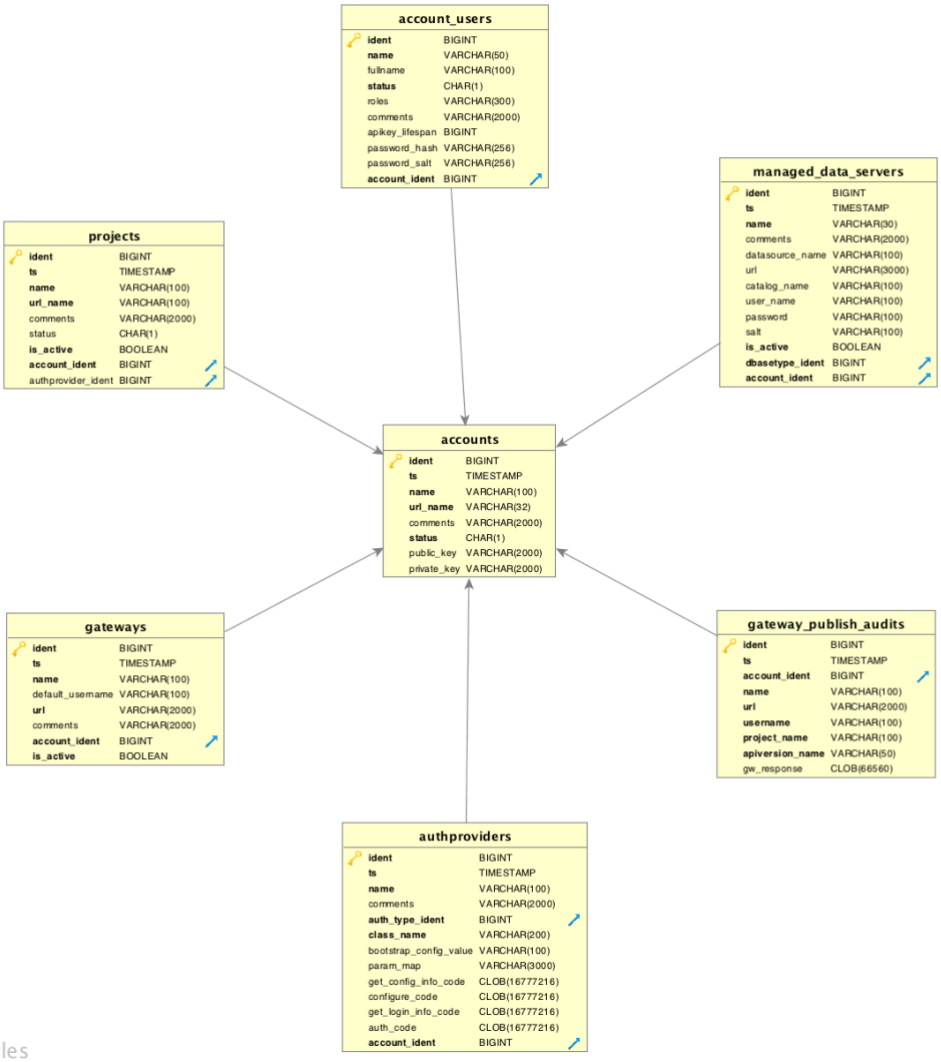
</td>
</tr>
</table>

TeamSpace (aka Accounts)

A new REST server creates a root object to which all rest objects will be linked called ‘accounts’ – known as TeamSpace. The 2 default types of accounts are the admin account url fragement ‘***abl’*** and the other url fragment is ‘***default’***. User API projects may be linked to the ‘default’ TeamSpace. When making a request to the admin ‘***abl’*** account the URL will look something like this: http://server[:port]/rest/abl/admin/v2/[endpoint] See Documentation. When making a request to a user TeamSpace endpoint it will look something like this: (TeamSpace = default) http://server[:port]/rest/default/[api_url]/[version]/[endpoint].

Note: the ‘default’ URL fragment name can be renamed or new user TeamSpaces can be created for multi-tenant development. See TeamSpace command.

	Column Heading	Column Heading
name	System TeamSpace	Default TeamSpace
url_name	abl	Default
status	A	A
Ident	1	2000


account_users
ident BIGINT
name VARCHAR(50)
fullname VARCHAR(100)
status CHAR(1)
roles VARCHAR(300)
comments VARCHAR(2000)
apikey_lifespan BIGINT
password_hash VARCHAR(256)
password_salt VARCHAR(256)
account_ident BIGINT
managed_data_servers
ident BIGINT
ts TIMESTAMP
name VARCHAR(30)
comments VARCHAR(2000)
datasource_name VARCHAR(100)
url VARCHAR(3000)
catalog_name VARCHAR(100)
user_name VARCHAR(100)
password VARCHAR(100)
salt VARCHAR(100)
is_active BOOLEAN
dbasetype_ident BIGINT
account_ident BIGINT
projects
ident BIGINT
ts TIMESTAMP
name VARCHAR(100)
url_name VARCHAR(100)
comments VARCHAR(2000)
status CHAR(1)
is_active BOOLEAN
account_ident BIGINT
authprovider_ident BIGINT
accounts
ident BIGINT
ts TIMESTAMP
name VARCHAR(100)
url_name VARCHAR(32)
comments VARCHAR(2000)
status CHAR(1)
public_key VARCHAR(2000)
private_key VARCHAR(2000)
gateways
ident BIGINT
ts TIMESTAMP
name VARCHAR(100)
default_username VARCHAR(100)
url VARCHAR(2000)
comments VARCHAR(2000)
account_ident BIGINT
is_active BOOLEAN
gateway_publish_audits
ident BIGINT
ts TIMESTAMP
account_ident BIGINT
name VARCHAR(100)
url VARCHAR(2000)
username VARCHAR(100)
project_name VARCHAR(100)
apiversion_name VARCHAR(50)
gw_response CLOB(66560)
authproviders
ident BIGINT
ts TIMESTAMP
name VARCHAR(100)
comments VARCHAR(2000)
auth_type_ident BIGINT
class_name VARCHAR(200)
bootstrap_config_value VARCHAR(100)
param_map VARCHAR(3000)
get_config_info_code CLOB(16777216)
configure_code CLOB(16777216)
get_login_info_code CLOB(16777216)
auth_code CLOB(16777216)
account_ident BIGINT

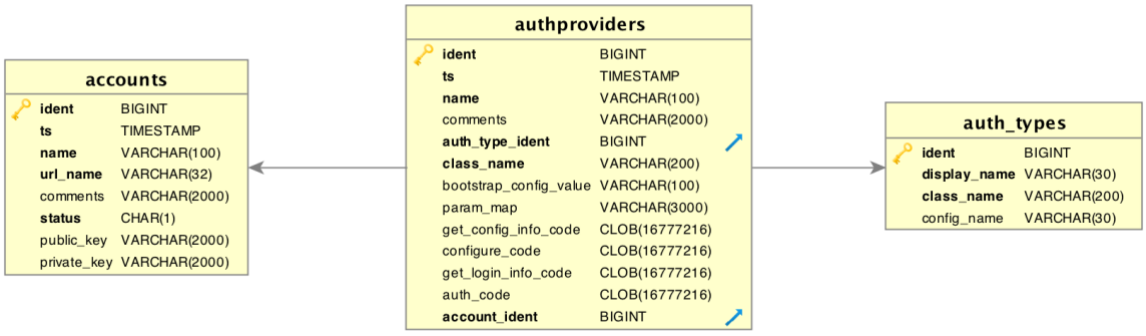
Help

```
lacadmin teamspace -h

  Usage: teamspace [options] <list|exportRepos>

  List TeamSpace content for current server or exportRepos the entire API
contents.

  Options:

   -f, --file [file]                          optional:: for source extract, the name of a file to read from/save to,
   --format [json|zip]                        optional: for import/export, this sets the output type of the export
   --passwordstyle [skip|encrypted|plaintext] optional: for export, sets the password style of exported API data sources
   --librarystyle [emit_all|in_use_only]      optional: for export, sets the library style  (default: emit_all)
   --apioptionsstyle [emit_all|skip_default]  optional: for export, sets the api options (default: emit_all)
   -v, --verbose                              optional: used by list to display each API in detailed export/import
   -h, --help                                 output usage information
```

The list command will show all the active and inactive teamspaces (aka accounts). The 'exportRepos' command will export the entire content of the connected teamspace as a zip or json file. The optional properties are the same as the 'lacadmin api' export command.

Authentication Providers

Authentication providers can be either the built-in default or a user provided JavaScript library. An API project is associated with a custom JavaScript, an HTTP Auth provider, or the default built-in authentication library. The authprovider will handle the login validation using a username/password and return a list of valid roles for the specified logon. If you write a custom authprovider (JavaScript or HTTP) – if these make use of Java JAR files or JavaScript Libraries they must be loaded before attempting to configure a new authprovider. See Create Custom Authprovider Using JavaScript Documentation.



Help

$lacadmin authprovider --help

```
Usage: authprovider [options] <list|create|linkProject|delete|export|import>

Administer authentication providers for an account.

Options:

  -h, --help                  output usage information
  --ident [ident]             The ident of the auth provider
  --project_ident [ident]     The project ident used to link this auth provider
  --name [name]               Name of auth provider
  --createFunction [bootstrap]  Name for Create Function
  --paramMap [map]            Map of auth provider settings
  --comments [comment]        Comment on auth provider
  --file [fileName]           [Optional] Name of file to Import/Export
```

List

Most of the lacadmin commands offer a list service to give a quick report on the contents of a specific admin table. This is just a quick report and some of the details may not be printed. See Appendix A for a sample script to report (list) all the contents of your project.

$lacadmin authprovider list

```
All authentication providers
Ident  Name                   createFunction  ParamMap                                                                                                                                                    Comm
ents
-----  ----------------------  -------------  -----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--  --------------------------------------
2000   AuthProviderFromDB      create         logonApiKey=Bzn8jVyfOTiIpW6UQCgy,loginBaseURL=http://localhost:8080/rest/default/b2bderbynw/v1/nw:Employees,l
oginGroupURL=http://localhost:8080/rest/default/derby/v1/nw:Region  Uses Get Employees for REST Validation
1000   Built-in authentication  null          datasource=AdminDB
```

Import/Export

Most ***lacadmin*** commands will offer an import/export service to retrieve specific database content in JSON file format. If the double-dash file (--file) command is used, then files are written/read to and from the file (or directory name and file). If omitted, the contents will be written to the console. The export format is shown below.

```
$lacadmin authprovider export
```


```
{
  "name": "Built-in authentication",
  "comments": null,
  "auth_type_ident": 1,
  "class_name": "com.kahuna.server.auth.db.DefaultAuthProvider",
  "bootstrap_config_value": null,
  "param_map": "\"datasource\"=\"AdminDB\"",
  "get_config_info_code": null,
  "configure_code": null,
  "get_login_info_code": null,
  "auth_code": null,
  "account_ident": null
  }
```


Delete

Some of the lacadmin commands offer a delete service for a specific row and require a double-dash (--ident [9999] or aut_name) as part of the command. To see a list of idents or auth_names, simply use the 'list' command. Note: You can not delete an authprovider if it is used by a specific project. Instead of an ident, you can use the auth_name parameter (--auth_name "MyAuthProvider").

LinkProject

Since an authprovider is a definition of instructions linking your authentication service to the definition, the command linkproject requires a double-dash (--project_ident [9999]) to instruct the api to use this specific authprovider. To get a list of api projects, use the syntax **(`$lacadmin api list`)**. If the authprovider is already loaded – when an api is imported it will be matched by name and the link may not be needed.

Auth Types

Auth Providers can either use the built-in default authprovider or a custom Java Script library. By default – the lacadmin command will use auth_type_ident = 2.

1	Default Auth Provider	com.kahuna.server.auth.db.DefaultAuthProvider
2	JavaScript Auth Provider	com.kahuna.server.auth.JavaScriptAuthProvider
3	HTTP Auth Provider	com.kahuna.server.auth.HttpAuthProvider
4	TeamSpace Auth Provider	com.kahuna.server.auth.db.TeamSpaceAuthProvider

The following sections cover account level meta data (that is, they are linked to the account and not to a specific project within an account); Gateways, Managed Servers, Auth Providers, Server Info, Libraries, and Licenses.

Class_name

This is the internal name used by the CA Live API Creator and cannot be changed

(com.kahuna.server.auth.JavaScriptAuthProvider)

ParamMap

This is a comma separated list of valid values used by the authentication provider .

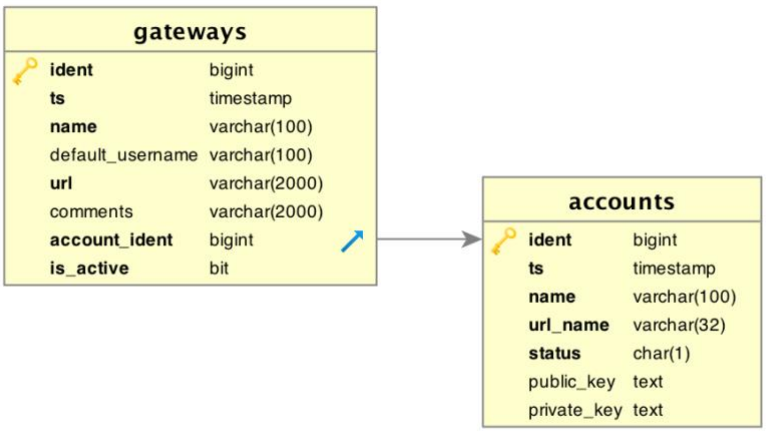
CreateFunction

This is the JavaScript function name used to instantiate the authentication provider.

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admn:authproviders

Gateways

The CA Live API Creator can publish a specific API Project Swagger 2.0 doc to one or more CA API Gateway host locations. You must provide the host URL, username and password, and the URL fragment for the specific project to publish the API Swagger. See Integrate with CA Gateway documentation.




```
$lacadmin gateway --help
```

```
Usage: gateway [options] <list|create|delete|import|export|publish>

Publish Swagger document for current project to CA Gateway.

Options:

  -h, --help              output usage information
  --ident [ident]         The ident for the saved gateway definition
  --name [name]           The name for the gateway definition
  --username [name]       The username for the gateway
  --password [password]   The gateway password.
  --hostname [server]     The gateway server hostname
                                          https://myserver:8443/lacman/1.0/publish
  --apiversion [version]  The API version of the swagger document
  --url_name [name]       The API url fragment name (use project list)
  --comments [comments]   The gateway definition comments
  --file [fileName]       [Optional] Name of file to import/export (if not provided
                                          stdin/stdout used for export)
```

Create

The create command will simply store the passed in parameters in the gateway admin database. These can be later exported or you can use the import command to store these definitions.

Publish

The command publish will take the parameters shown above and publish a specific project Swagger 2.0 documentation to a CA Gateway (hostname). To see a list of url names use **(`$lacadmin project list`)**. To see a list of defined gateways use ***lacadmin gateway list***.

```
$lacadmin gateway publish --ident 1 --url_name demo --username demo --password password --
apiversion v1
```

Export/Import

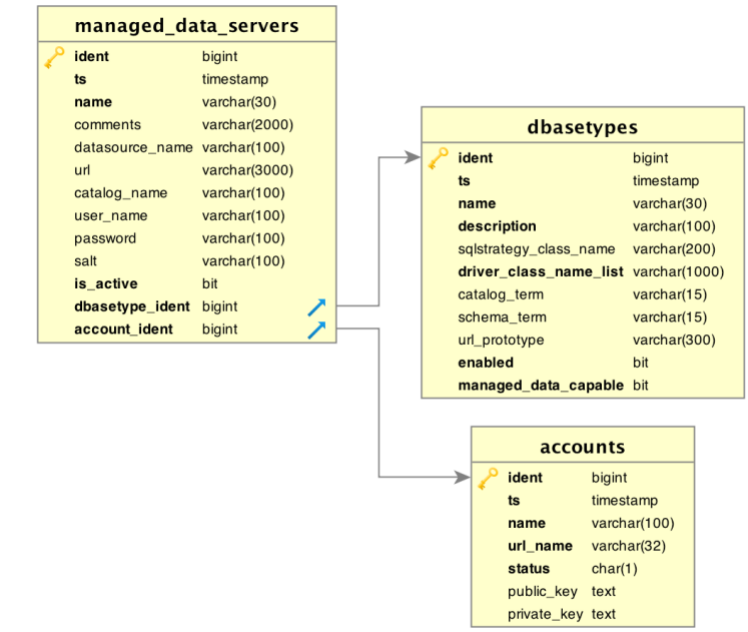
```
$lacadmin gateway export
[
 {
   "ts": "2017-01-02T17:11:34+00:00",
   "name": "New CA API Gateway",
   "default_username": "test",
   "url": "http://lod19:7890/publish",
   "comments": "Test Publish API to Gateway",
   "is_active": true
 }
]
```

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/gateways

Managed Data Servers

A Managed Data Server is based on a specific admin database table ***dbasetypes*** (e.g. MySQL, Postgresql, Oracle, MS Sql Servder, or Derby). Once a managed server is defined a new database and datasource can be created using this prototype definition. The API wizard in Live API Creator offers 'code first' or 'model first' options to use your new definition. Or you can use the 'add' button in the datasource itself to create a new editable connection.

See Managed Data Server Administration documentation.




```
$lacadmin managedserver --help
```

Usage: managedserver [options] <list|create|update|delete|import|export>

Administer a managed data server (used by @databases to create datasources).

Options:

-h, --help	output usage information
--server_name [name]	The name of the datasource connection
--ident [ident]	For delete or reload, the ident of the managed data server
--dbasetype [dbasetype]	The type of the managed data server connection, can be mysql, derby, postgres
--catalog_name [catalog_name]	The catalog in the managed data server
--user_name [user_name]	The name of the managed data server user
--password [password]	The password of the managed data server user
--url [url]	The JDBC URL for the managed data server
--comments [comment]	This is the comment for this managed data server
--active [true|false]	This marks the managed data server active or inactive
--file [file]	Optional: for import/export, the name of a file to read from/save

List

```
$lacadmin managedserver list
Managed Data Server(s)
Ident  Name                    Type     Active  Catalog  User            URL                              Comments
-----  ----------------------  -------  ------  -------  --------------  -------------------------------  --------------
2000   Demo managed data server  unknown  true    null     DEMO_MANAGED_DATA  jdbc:derby:directory:ManagedData;create=
```

Export/Import

This is a sample of the JSON content of a managed derby server (dbasetype_ident = 17).

```
$lacadmin managedserver export
[
 {
   "ts": "2016-12-03T06:13:39.457",
   "name": "Demo managed data server",
   "comments": "This is created as part of the evaluation package",
   "datasource_name": null,
   "url": "jdbc:derby:directory:ManagedData;create=true",
   "catalog_name": null,
   "user_name": "DEMO_MANAGED_DATA",
   "password": null,
   "salt": null,
   "is_active": true,
   "dbasetype_ident": 17,
   "account_ident": 1000,
   "project_ident": null
 }
]
```

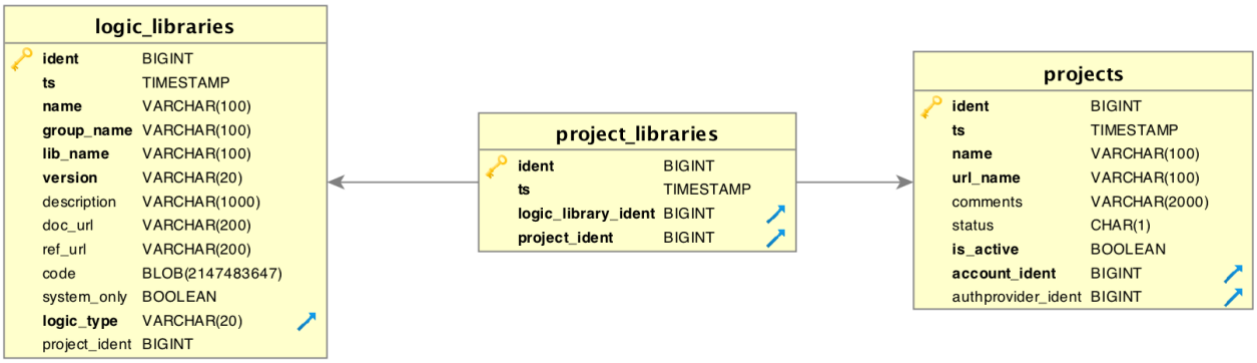
Update

The update command will replace specific values for a managed server using the double-dash (e.g. --ident [9999], --password).

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:managed_data_servers

Logic Libraries (JavaScript)

CA Live API Creator allows users to create or import JavaScript libraries to use in reactive logic rules, resource definitions, functions, and events. As of 4.0, libraries are now stored at the individual api level and will not be shared.



API Use Command

The lacadmin command line will more than likely be applied to a specific API projects (**`$lacadmin api list`**). This will allow all future commands to use the current api (and avoid using the double-dash project_ident on certain project level commands.)

```
$lacadmin api use –url_name demo
```

Project Libraries

In 4.0 and earlier, a single logic library was linked to one or more user API projects. A JavaScript library can be added to the system and used in reactive logic (rules), JavaScript resources, functions, and request/response events as well as an authentication provider. Note: If the library is loaded for the first time, the use flag in CA Live API Creator will load this library into memory. Each API project will have a unique and single instance of a logic library.

```
$lacadmin libraries --help
```

Usage: libraries [options] <list|create|update|delete|export|import>

Administer JavaScript libraries for an account.

Options:

```
-h, --help                  output usage information
--ident [ident]             The ident of the library
--project_ident [projectId]  The project ident that this library will be marked as
                                                    used
--name [name]               Name of library
--libtype [type]            Type of Library javascript
--ver [version]             Version # of Library
--short_name [shortname]    Short Name
--docUrl [docurl]           Documentation URL
--refUrl [refurl]           Reference URL
--linkProject               Link the imported library to the current project
--comments [comment]        Comment on Library
--file [fileName]           [Optional] Name of JS file to import/export
```

List

Note that libraries with ident values over 999 are user defined.

`$lacadmin libraries list`

All Libraries

Ident	Name	Version	Short Name	Type	Comments
2002	AdminAuthProvider	1.0	AdmAuth	javascript	for
2000	B2BAuthProvider	1.0	CustomAuth	javascript	imported in install scripts from file b2b/scs/p...
2001	B2BLibrary	1.0	b2bLib	javascript	imported in install scripts, from file: b2b/scs...
511	Moment-Timezone.js	0.5.3	moment-timezone	javascript	Parse and display dates in any timezone.
500	Moment.js (with Locales)	2.12.0	moment	javascript	A JavaScript date library for parsing, validati...
507	Numeral.js	1.5.3	Numeral.js	javascript	A JavaScript library for formatting and manipul...
503	ParsedURL - URL parser	1.0	ParsedURL	javascript	A simple JavaScript class/library for parsing/m...
504	Underscore.js Library	1.8.3	underscore	javascript	Underscore is a JavaScript library that provide...
502	jkl XML parser	0.22	jkl	javascript	A pure-JS XML parser.
501	json2	20150503	json2	javascript	A pure-JS implementation of the JSON interchang...

Export/Import

The library JavaScript export code is encoded as either hex (0x…) or Base64 (b64:) - If you create a new library, you can use plain JavaScript as input text.

```
$lacadmin libraries export --ident 2000
[
 {
  "name": "B2BAuthProvider",
  "group_name": "CustomAuth",
  "lib_name": "CustomAuth",
  "version": "1.0",
  "description": "imported in install scripts from file b2b/scs/projects/sharedlibs/src/AuthProviderFromDB.js",
  "doc_url": null,
  "ref_url": null,
  "code": "0x2f2f20437573746f6d206175…….truncated hex ….",
  "system_only": false,
  "logic_type": "javascript"
 }
]
```

Link Project

If you are creating a library and want to link it to the current project, include the double-dash (--**`linkProject`**) flag if this library will be marked as used in the current or selected project (--project_ident [9999]).

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:logic_libraries

API (aka projects)

The API default account may have one or more user API projects. API Projects contain the detail contents and the url_name fragment used by all REST service endpoints. The url_name can be changed but caution should be used since this may impact code, resources, or rules that referenced the original url_name. See API's project URL documentation.

Note: The ***lacadmin project*** has been **deprecated** in 4.1 – however it is still available for users running prior releases. The project command will export JSON in the original format not the new

enhanced format. 4.1 API exports cannot be imported into 4.0 LAC. However, the project endpoint will work continue to work with 4.1 using the older JSON format.

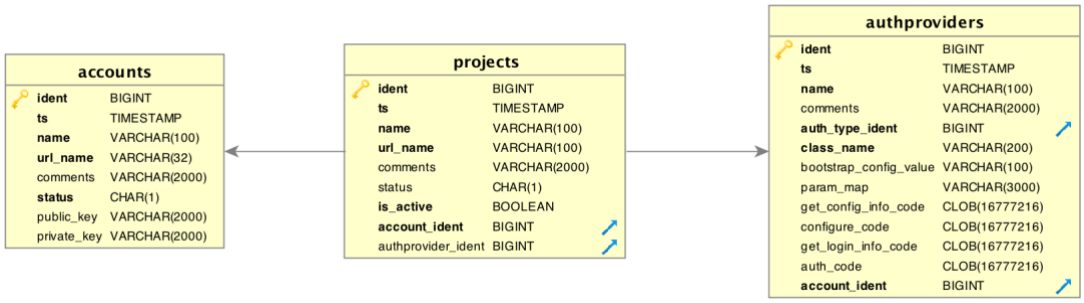
*http://[SERVER:[PORT]/[war]/[**rest**|**data**]/[abl|default]/[url_name]/[version]/[ENDPOINT]*

Note 1: the project is associated with an Auth Provider instance.

Note 2: User API [***account***] is always ***default***. Admin [***account***] is 'abl'.

Note 3: The term '***rest'*** in the URL is reserved for all requests. A special endpoint called '***data'*** is used when dealing with CLOB, BLOB, and Image endpoints.

Note 4: '***version'*** will default to 'v1' for new projects.




```
$lacadmin api --help
Usage: api [options] <list|create|update|delete|use|import|export|extract>

Administer API. Actions are: list, create, update, delete, use, import, export, extract

Options:

  --ident [ident]                                                  The ident of the specific project (use $lacadmin api list)
  --api_name [name]                                                The name of the API
  --url_name [name]                                                The url fragment name of the API
  --status [status]                                                optional: create or update the status of the API, can be A (for Active) or I for (Inactive)
  --authprovider [ident]                                           optional: create or update the ident of the authentication provider for the API
  --comments [comments]                                            optional: create or update a description of the API
  -d, --directory [directory]                                      Required for extract, the name of a directory to extract ZIP files
  -f, --file [file]                                                optional: for import/export, the name of a file to read from/save to, if unspecified, use stdin/stdout
  --format [json|zip]                                              optional: for import/export, this sets the output type of the export default: zip
  --section [name]                                                 (optional) The section of the API you wish to export (e.g. resources, functions, datasources)
  --section_filter [filter]                                        (optional) The section filter of the API you wish to export (name=foo&version=v1)
  --namecollision [fail|rename_new|replace_existing|disable_and_rename_existing]  optional: for import, determines how to handle existing API (default: rename_new)
  --errorhandling [standard|fail_on_warning|best_efforts]          optional: for import, sets the error level response hanling (default: standard
  --passwordstyle [skip|encrypted|plaintext]                       optional: for export, sets the password style of exported API datasources (default: skip)
  --librarystyle [emit_all|in_use_only]                            optional: for export, sets the library style  (default: emit_all)
  --apioptionsstyle [emit_all|skip_default]                        optional: for export, sets the api options (default: emit_all)
  --synchronize [true|false]                                       optional: Used by extract & synchronize zip file with directory folder (default: false)
  -v, --verbose                                                    optional: whether to display detailed results, or just a summary
  -h, --help                                                       output usage information
```

DevOps Note: The JSON file is a complete definition of the entire API include rules, functions, resources, security settings, events, and libraries. If you intend to migrate this api to another server then you will need to first export any custom auth providers first (these must be imported into the

new server first). Also – The Live API Creator project export may not have exported Datasource (aka database) passwords – these may need to be updated manually use lacadmin or other services.

List

```
$lacadmin API list
```

```
All API's
Ident  Name            Enabled  URL Name       Comments
-----  --------------- -------  -------------  ----------------------------------------
2002   B2B Derby Pavlov  true     b2bderbypavlov  B2B Demo for Supplier Pavlov.  LAC
2003   B2B Northwind     true     b2bderbynw      Business to Business (B2B) Sample. Explore
2000   Demo              true     demo            Welcome examples.
2001   Sample            true     sample          Complex examples, per Logic Tutorial.  See
```

Create

The create command will allow you to create an empty API project with a given name, url_name fragment, and optional authprovider ident (to get a list of auth providers use ***lacadmin authprovider list***).

```
$lacadmin api create --api_name <name>
    --url_name <url_name> [--status <A|I>]
    [--authprovider <ident>]
    [--comments <comments>] [--verbose]
```

Use

The command 'lacadmin project use' followed by either the double-dash ident (--ident [9999]) or the project url fragment name (--url_name [URL name fragment]) will store this with the current login information and allow all future lacadmin commands to work with this selected project. Any lacadmin command can override this by providing a double-dash --project_ident [9999] or by switching projects with the use command.

```
$lacadmin api use [--api_name '<name>' | --url_name <url_name>]
```

Export API

Any API Project can be exported as an entire JSON file, this includes data sources, libraries, functions, resources, rules, security settings, property configurations, application settings, etc.

Note 1: Datasource passwords may not have been exported and may need to be manually fixed up using command line or using the CA Live API Creator GUI.

```
$lacadmin api export
         [ --api_name <name> | --url_name <url_name>]
         [ --file <filename> ]
         [ --format [zip|json] ]
         [ --passwordstytle [skip|encrypted|plaintext] ]
         [ --librarystyle [emit_all|in_use_only]  ]
         [ --apioptionsstyle [emit_all|skip_default] ]
         [ --verbose]
```

`Tip` – You can always use the command **`lacadmin api list`** to see the current API names, idents, and url_fragment names.

Export API JSON File Sample (compact) 4.0 and earlier


```
[
  {
    "ident": 2002,
    "ts": "2016-03-04T12:43:43.000Z",
    "exportVersion": "v4",
    "adminSchemaVersion": "20160303",
    "name": "Banking Demo",
    "url_name": "banking",
    "comments": "1. Deploy your SQL to a database\n2. Change your Data Source database connections\n3.
    "status": null,
    "is_active": true,
    "account_ident": 1000,
    "authprovider_ident": 1000,
    "@metadata": {…},
    "Topics": […],
    "Rules": […],
    "ApiVersions": […],
    "DbaseSchemas": […],
    "ProjectLibraries": […],
    "ProjectOptions": […],
    "ApiKeys": […],
    "Roles": […],
    "Users": […],
    "Handlers": […],
    "EventHandlers": […],
    "Relationships": […],
    "Applications": […],
    "NamedFilters": […],
    "NamedSorts": […]
  }
]
```


As of 4.1 the JSON file format is expanded to show file contents using {f} designation and directory hierarchy using {d}. The organization is designed to make merging easier between 2

JSON files. If the ZIP format is selected for export, the file will be identical, but directories and files will be in exploded into their proper places and can easily be checked into your version/source control system.

Repository Root

Live API Creator will store the in-memory contents on the local file system. The organization will be divided between the /system directory and /teamspaces.

System directory

This will contain the information used by the admin account to bootstrap the server. It holds the EULA and license information as well as key definition files to define teamspace authentication. One file to note is 'lacignore.json' that can define which files and directories to ignore (useful for source control intergration).

```
.
├── adminMetaCache.json
├── apioptions.json
├── authProviderName.json
├── auth_providers
│   ├── ReadMe.md
│   └── TeamSpace$0020Auth$0020Provider
├── authtokens
│   ├── ReadMe.md
│   ├── Temp$0020AuthToken$0020for$0020admin$0020$005b2018-07…d.json
├── data_source_providers
│   ├── Mongo
│   ├── Prototype
│   └── ReadMe.md
├── data_sources
│   └── ApiKey.json
├── eula.json
├── gateways
│   └── ReadMe.md
├── lacignore.json
├── libraries
│   ├── ReadMe.md
│   └── SystemLibraryUsage.json
├── license.json
├── serverIdent.json
```

```
└── teamspace_users
    ├── ReadMe.md
    └── sa.json
```

Teamspaces Directory

Each teamspace (e.g. default) will contain one or more apis, auth providers, gateways, and manged server definitions. Note that the teampsace.json file and teamspace-users folder contain the specific information used to define the teamspace itself. If you clone this directory, remember to change the teampsace.json file and change the user entries as well.

```
.
└── default
    ├── ReadMe.md
    ├── apis
    │   ├── b2bderbynw
    │   ├── b2bderbypavlov
    │   ├── conf-management
    │   ├── conf-offer
    │   ├── demo
    │   ├── demo_db2luw
    │   ├── demo_derby
    │   ├── demo_mysql
    │   ├── demo_oracle
    │   ├── demo_postgresql
    │   ├── demo_sqlserver
    │   ├── jms
    │   ├── kafka
    │   ├── mqtt
    │   ├── rabbitmq
```

```
│   ├── sample
│   ├── sample_db2luw
│   ├── sample_mysql
│   ├── sample_oracle
│   ├── sample_postgresql
│   ├── sample_sqlserver
├── auth_providers
│   ├── Built-in$0020authentication
│   ├── ReadMe.md
│   └── SimpleLDAP
├── gateways
│   └── ReadMe.md
├── managed_servers
│   └── ReadMe.md
├── teamspace.json
└── teamspace_users
    ├── ReadMe.md
    ├── admin.json
```

Import API

The project import will take an existing export JSON file and create a new project name. Note: The import command will make the newly imported API the active API. If the API name already exists, it will append a timestamp to the name. The optional argument –***namecollision*** default is ***rename_existing***. The import can detect 4.0 formats and will use the original endpoint, starting in 4.1 can import will accept both JSON and ZIP file formats. Note: Do not change the directory names.

```
$lacadmin api import --file <filename[.json|.zip]>
        [ --namecollision
             [fail|rename_new|replace_existing|disable_and_rename_existing] ]
        [ --errorhandling standard ]
        [ --verbose]
```

DevOps Note – Remember, the export may not have Datasource passwords. You can use the command line lacadmin datasource update the selected passwords.

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:projects

Section

The export option can select a specific section of the API and format the output as either ZIP or JSON. This output will be in the new format (JSON) or can be exported as a ZIP (use ***--format zip***). These partial sections can be merged into a master JSON file or merged into a source control directory.

```
EXPORT PROJECT=demo
mkdir temp
$lacadmin api export --section api --file temp/api.json --format json
$lacadmin api export --file temp/API.zip --format zip
$lacadmin api export --section connections --file temp/connections.json
$lacadmin api export --section custom_endpoints --file temp/custom_endpoints.json
$lacadmin api export --section listeners --file temp/listeners.json
$lacadmin api export --section relationships --file temp/relationships.json
$lacadmin api export --section relationships --file temp/relationships.json
$lacadmin api export --section data_sources --file temp/datasources.json
$lacadmin api export --section filters --file temp/filters.json
$lacadmin api export --section functions --file temp/functions.json
$lacadmin api export --section libraries --file temp/libraries.json
$lacadmin api export --section request_events --file temp/request_events.json
$lacadmin api export --section sorts --file temp/sorts.json
$lacadmin api export --section timers --file temp/timers.json
$lacadmin api export --section topics --file temp/topics.json
$lacadmin api export --section security  --file temp/security.json
$lacadmin api export --section security.authtokens  --file temp/authtokens.json
$lacadmin api export --section security.roles --file temp/roles.json
$lacadmin api export --section security.users --file temp/users.json
$lacadmin api export --section resources --file temp/resources.json
$lacadmin api export --section rules --file temp/rules.json
$lacadmin api export --section apioptions --file temp/apioptions.json
```

Section with Filter

The export command will allow you to export a specific section and find a specific object by name using the ***section_filter*** option.

```
EXPORT PROJECT=demo
mkdir temp2
$lacadmin api export -section api --file temp2/api.json --format json --url_name
$PROJECT
$lacadmin api export --section connections --section_filter "name=MQTTConn" --file
temp2/connections.json
$lacadmin api export --section custom_endpoints --section_filter "name=endpoint" -
-file temp2/custom_endpoints.json
$lacadmin api export --section listeners --section_filter "name=START" --file
temp2/listeners.json
$lacadmin api export --section relationships --section_filter
"parent_entity=demo:customer" --file temp2/relationships.json
$lacadmin api export --section data_sources --section_filter "prefix=demo" --file
temp2/datasources.json
$lacadmin api export --section filters --section_filter "name=UserFilter" --file
temp2/filters.json
$lacadmin api export --section functions --section_filter "name=testFunction" --
file temp2/functions.json
$lacadmin api export --section libraries --section_filter "name=b2bB2B" --file
temp2/libraries.json
$lacadmin api export --section request_events --section_filter
"name=ResponseEvent"  --file temp2/request_events.json
$lacadmin api export --section sorts --section_filter "name=UserSort" --file
temp2/sorts.json
$lacadmin api export --section timers --section_filter "name=New Timer" --file
temp2/timers.json
$lacadmin api export --section topics --section_filter "name=Audit Orders"  --file
temp2/topics.json
$lacadmin api export --section security.authtokens  --section_filter "name=Admin
key" --file temp2/authtokens.json
$lacadmin api export --section security.roles --section_filter "name=Read only" --
file temp2/roles.json
$lacadmin api export --section security.users --section_filter "name=guest" --file
temp2/users.json
$lacadmin api export --section resources --section_filter
"name=AllCustomers&version=v1" --file temp2/resources.json
$lacadmin api export --section rules --section_filter
"name=sum_balance&prefix=demo"  --file temp2/rules.json
$lacadmin api export --section apioptions --section_filter "name=Force Binary Data
as an Object" --file temp2/apioptions.json
```

Extract

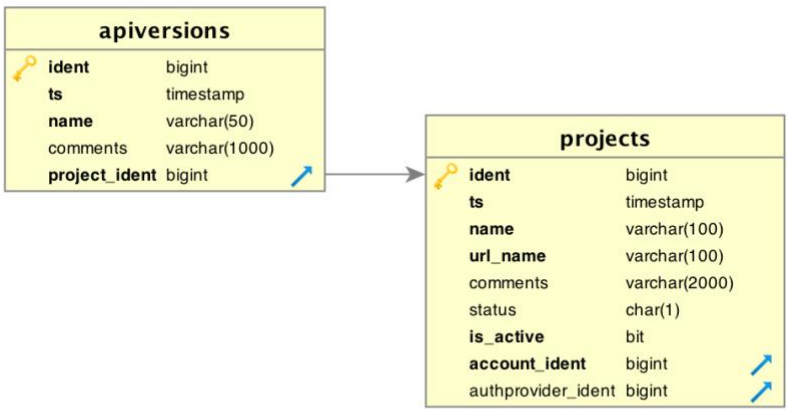
The ZIP file format will expand into a hierarchy of directories and files. It may be desired to synchronize the content of the new ZIP file with the directory. The synchronize flag will compare the ZIP file content and remove any files in the directory that are not found in the zip file and

replace (overwrite) the contents of the directory if the file/directory exists. In the 5.0 release, the LAC_REPOSITORY_ROOT location will already have this expanded view of your APIs.

```
$lacadmin api extract
        --file <filename.zip>
        --directory </tmp/path/>
        --synchronize [true|false]
```

API Versions

Resource objects can be versioned and are children of the apiversion table. In the CA Live API Creator GUI – when a new API Version is created, all resources from the selected version will be cloned.



$lacadmin apiversion --help

Usage: apiversion [options] <list|export|import>

Administer API Versions for Resources for current project.

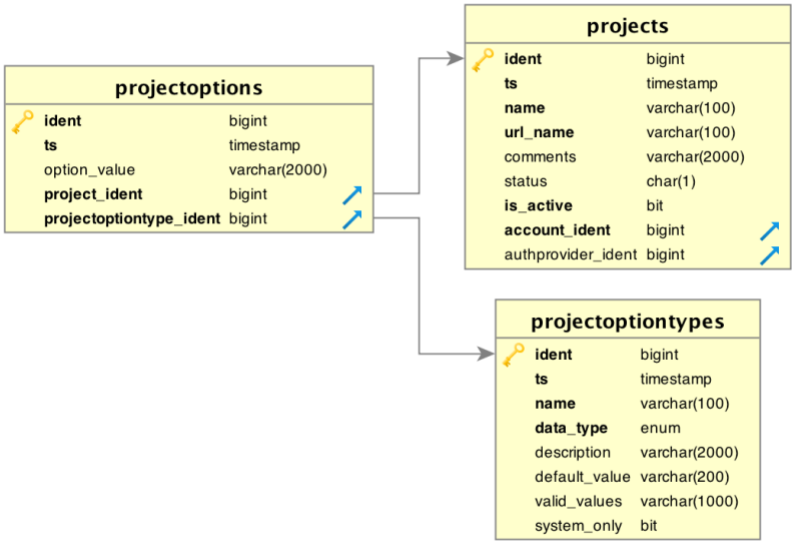
Options:

-h, --help output usage information
--project_ident [project_ident] The project ident that will be used
--file [fileName] [Optional] Name of file to import/export

Note: Creating a new Version using the command line will **not** clone your prior version resources. However, if you export your current resource, you can import it using the new version identifier.

API Options

These are the settings that control many of the internal functions and processing of REST requests like default pagesize, binary output, default response format, xml root, etc.



Use $**lacadmin apioptions list** for a specific sample project. These options control pagination limits, Swagger, image handling, enable basic authentication, etc. See Managing API Properties documentation.

```
$lacadmin apioptions --help
```

```
Usage: apioptions [options] <list|update|import|export>

Administer API project options for an account.

Options:

  -h, --help                output usage information
  --ident [ident]           The ident of the specific project settings object
  --option_value [value]    This is the value for the specific setting for the ident
```

--project_ident [project_ident] The project ident that will be marked as used

--file [fileName] [Optional] Name of file to settings for import/export

$lacadmin apioptions list

```
BANTY01MAC1453:~ banty01$ lacadmin apioptions list
API Project Option Settings
Ident  Project  Name                                 Value
-----  -------  -----------------------------------  ------------------------------------------------------------------
2139   2008     Aggregate Default Override           false
2152   2008     Allow Swagger without authentication true
2154   2008     Binary Output Encoding               base64
2143   2008     Checksum Size Limit                  2000
2147   2008     Chunk Size Default                   20
2153   2008     Enable HTTP Basic Authentication     true
2155   2008     Force Binary Data as an Object       true
2141   2008     HTTPS only                           false
2144   2008     Inline Limit Default                 6000
2146   2008     Maximum Page Size                    5000
2142   2008     Metadata name                        @metadata
2145   2008     Page Size Default                    20
2150   2008     Permit Authorization parameter in URL true
2149   2008     Stored Procedure Inline Limit        2000
2148   2008     Stored Procedure Row Limit           100
2151   2008     Tech docs URL                        http://ca-doc.espressologic.com/docs/tutorial/business-logic-demo
2140   2008     Type base URI                        urn:caliveapicreator:examples:demo:

# settings: 17
```

Export/Import

The import must be in the context of a specific project.

$lacadmin apioptions export --ident 2000


```
[
 {
  "ts": "2016-12-03T06:13:25.454",
  "option_value": "false",
  "projectoptiontype_ident": 1,
  "ProjectOptionTypes": {
   "ident": 1,
   "ts": "2016-12-03T06:12:05.657",
   "name": "Aggregate Default Override",
   "data_type": "boolean",
   "description": "Controls whether setting/altering aggregates is overridden (default), or raises an exception.",
   "default_value": "false",
   "valid_values": null,
   "system_only": false
```

```
  }
 }
]
```

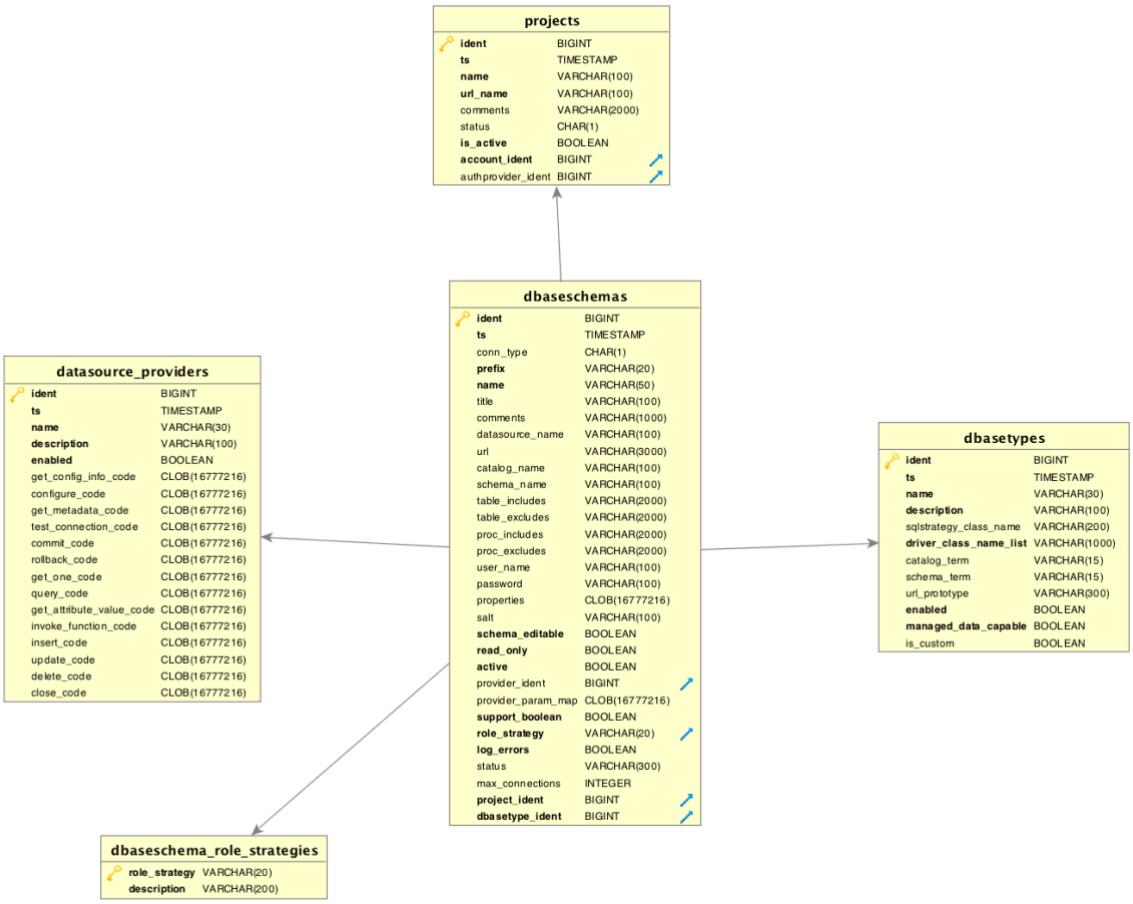
REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:projectoptions

Data Sources

The connection wizard will list the available databases supported by CA Live API Creator. The wizard will use the meta table '***admin:dbasetypes'*** for a list of valid and enabled JDBC drivers. The wizard will also provide some input fields for the specific database connection. An API project may have one or more data source connections. This is a list of project dependent SQL database datasources. Using the command line, you can update the password for a specific Datasource prefix:

```
$lacadmin datasource update –prefix demo –password Password1
```

The prefix allows



multiple datasources to be used in a single API project. See Documentation.

```
$lacadmin datasource --help
```

Usage: datasource [options] <list|create|createDatabase|update|delete|import|reload|export>

Administer datasources within a project.

Options:

-h, --help	output usage information
--db_name [name]	The name of the datasource connection
--ident [ident]	For delete or reload, the ident of the datasource
--prefix [prefix]	The prefix of the datasource connection

--dbasetype [dbasetype]	The type of the datasource connection, can be mysql, oracle, sqlserver, derby, postgresql, db2luw, csv, hbase, sap, salesforce, db2zos, sqlserverazure
--catalog_name [catalog_name]	The catalog in the datasource
--schema_name [schema_name]	The schema in the datasource
--user_name [user_name]	The name of the datasource user
--password [password]	The password of the datasource user
--url [url]	The JDBC URL for the datasource
--active [true|false]	This marks the datasource active or inactive
--project_ident [ident]	The ident of a project, (if other than the current project
--managedserver_ident [managedserver_ident]	The managed server ident used with command createDatabase (creates both database and datasource)
--file [file]	Optional: for import/export, the name of a file to read

List

```
$lacadmin datasource list
```

Datasources

Name	Prefix	Type	Active	isEditable	Catalog	Schema	User	URL	Comments
Demo	demo	unknown	true	false	null	"DEMO"	"DEMO"	jdbc:derby:directory:/Users/BANTY01/derbytest/Demo	
Finance	finance	unknown	true	false	null	"FINANCE"	"FINANCE"	jdbc:derby:directory:/Users/BANTY01/derbytest/Finance	

datasources: 2

Export/Import

```
$lacadmin datasource export
[
 {
   "ts": "2016-12-03T06:13:25.361",
   "conn_type": null,
   "prefix": "demo",
   "name": "Demo",
   "comments": null,
   "datasource_name": null,
   "url": "jdbc:derby:Demo",
   "catalog_name": null,
```

```
  "schema_name": "\"DEMO\"",
  "table_includes": null,
  "table_excludes": null,
  "proc_includes": null,
  "proc_excludes": null,
  "user_name": "\"DEMO\"",
  "password": null,
  "salt": null,
  "schema_editable": false,
  "read_only": false,
  "active": true,
  "role_strategy": "deprecated",
  "log_errors": false,
  "status": null,
  "project_ident": null,
  "dbasetype_ident": 17
},
```

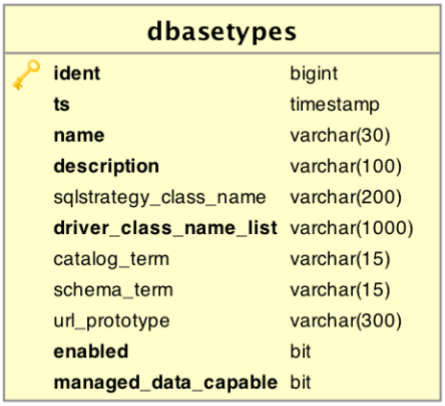
Create Database

If you have a Managed Data Source, the createDatabase command will allow a new database and Datasource to be created using this command option using the double dash --managed_server_ident switch.

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:dbaseschemas

Dbasetypes (internal)

This is an internal table with a list of all defined JDBC database types. If the enabled flag is set to false, this database will not appear in the connection wizard. Do not modify this table. Note: Some drivers have been marked as enabled = false and will not show up in the connection wizard. These are beta drivers or drivers that are no longer supported by CA Technologies.



Datasource Provider

The ability to define a custom datasource using JavaScript is supported using the dataprovider command. Note: There are multiple JS files that are stored on disk (/system/datasource_provider) that can be individually edited.

Help

```
$lacadmin dataprovider -h

  Usage: dataprovider [options] <list|delete|export|import>
```

```
  Administer Datasource Provider definitions.

  Options:

    --provider_name [name]  The Datasource Provider Name
    --ident [ident]         The ident of the specific provider
    -v, --verbose           optional: Display list of providers in detailed
export/import format
    --file [fileName]       optional: Name of file to import/export (if not
provided stdin/stdout used for export)
    -h, --help              output usage information
```

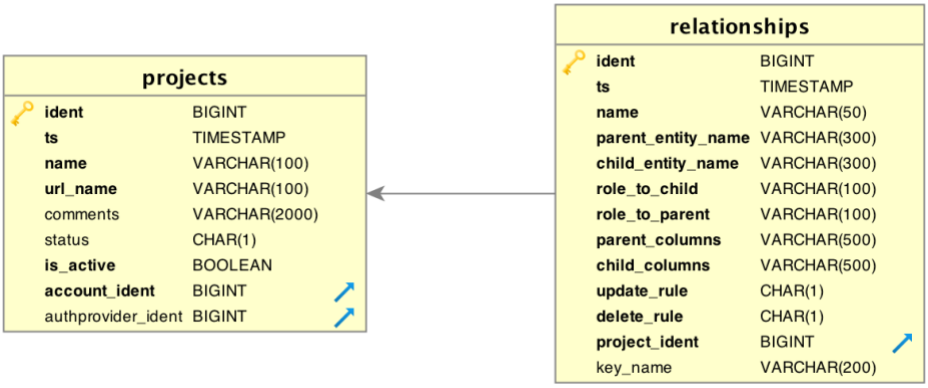
List

```
$lacadmin dataprovider list --verbose
Providers
Ident  Name         Enabled  Comments
-----  -----------  -------  ------------------------------------------------
2002   JDBCExample  true     JDBCExample derby Northwind datasource provider
2003   Mongo        true     Mongo datasource provider
2001   Prototype    true     Prototype datasource provider

# listener_providers: 3


$lacadmin dataprovider export --provider_name 'JDBCExample' --file  DATAPROVIDER_JDBCExample.json

#lacadmin dataprovider import --file DATAPROVIDER_JDBCExample.json

$lacadmin dataprovider export --provider_name 'Mongo' --file  DATAPROVIDER_Mongo.json

#lacadmin dataprovider import --file DATAPROVIDER_Mongo.json
```

Relationships (Virtual Foreign Keys)

This feature allows definitions to be made between any defined entities (tables and views) and entities across different databases. Users will select parent and child entity and corresponding parent/child attributes. These virtual relationships will be enforced during updates like foreign key definitions. One advantage is that these relationships can cross database type boundaries. For example, you can create a lookup (parent pick) using Derby on a DB2 zOS table. See Virtual

Foreign keys documentation.



```
$lacadmin relationship --help
```

Usage: relationship [options] <list|delete|export|import>

Administer Relationships (Virtual Keys) for current project.

Options:

-h, --help output usage information
--ident [ident] This is the ident of the relationship
--project_ident [project_ident] The project ident that will be used
--file [fileName] [Optional] Name of file to import/export

List

```
$lacadmin relationship list
Relationships
Ident  Parent Entity       Parent Cols      Child Entity             Child Cols       Role to Parent          Role to Child            Update Rule  Delete Rule
      Comments
-----  ------------------  ---------------  -----------------------  ---------------  ----------------------  -----------------------  -----------  -----------
----  --------
2091   demo:employee       "employee_id"    demo:employee_picture    "employee_id"    employee_picture        employee_pictureList     R            C
trwf
2093   demo:product        "product_number" demo:LineItem            "product_number" product                 LineItemList             R            R
zgkm
2094   demo:PurchaseOrder  "order_number"   demo:LineItem            "order_number"   lineitem_purchaseorder  LineItemList             R            C
nkvu
2089   demo:customer       "name"           demo:PurchaseOrder       "customer_name"  customer                PurchaseOrderList        R            C
mvwo
2090   demo:customer       "name"           finance:orders           "customer_name"  demoCustomer            financeOrders            R            R
zwcz
2095   demo:PurchaseOrder  "order_number"   demo:purchaseorder_audit "order_number"   purchaseorder_audit     purchaseorder_auditList  R            C
klbe
2092   demo:employee       "employee_id"    demo:PurchaseOrder       "salesrep_id"    salesrep                PurchaseOrderList        R            C
```

Export/Import

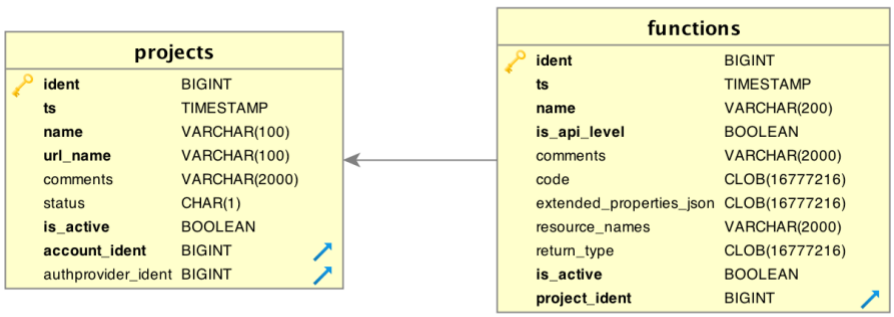
```
$lacadmin relationship export
[
 {
   "parent_entity_name": "demo:customer",
   "child_entity_name": "finance:orders",
   "role_to_child": "financeOrders",
   "role_to_parent": "demoCustomer",
   "parent_columns": "name",
   "child_columns": "customer_name",
   "update_rule": "R",
   "delete_rule": "R",
   "key_name": null
 }
]
```

Note: The name is a hash of the entity and attributes and is provides a unique index for the definition. The key_name is the original database foreign key constraint name.

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:relationships

Functions

A function is a pure JavaScript endpoint that can have optional named parameters (like a stored procedure). Functions can be either top level (a pure REST endpoint) or be tied to a specific entity (e.g. table or view) or normal (entity-backed) resource row. A function has a name, optional parameters, optional comma separated list of resource names, and JavaScript code. See Managing Functions documentation.



```
$lacadmin function --help
```

Usage: function [options] <list|delete|export|import>

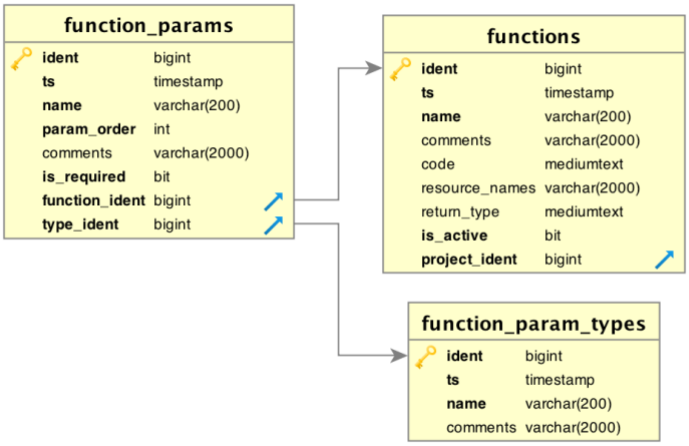
Administer Functions for current project.

Options:

-h, --help	output usage information
--ident [ident]	This is the ident of the function
--project_ident [project_ident]	The project ident that will be used
--file [fileName]	[Optional] Name of file to import/export

Function Parameters

Parameters are linked to the function and can have parameter type of 'string', number, or boolean.



Function List

```
$lacadmin function list
```

Functions:

Ident Name Parameters Resource Is Active Comments

---- ---------- ---------- ---------- --------- ----------

2002 giveRaise (number percentRaise) nw:Employees

EmployeesWithRaises true This is a skeleton function, you should most li...

2001 sendMail (string title) nw:Employees true A function associated with Resources (see tab a...

2004 testB2BAll () null true This is a skeleton function, you should most li...

2003 testB2BOrder () null true This is a skeleton function, you should most li...

2005 testB2BPavlovAuth () null true Test Sample

Export/Import

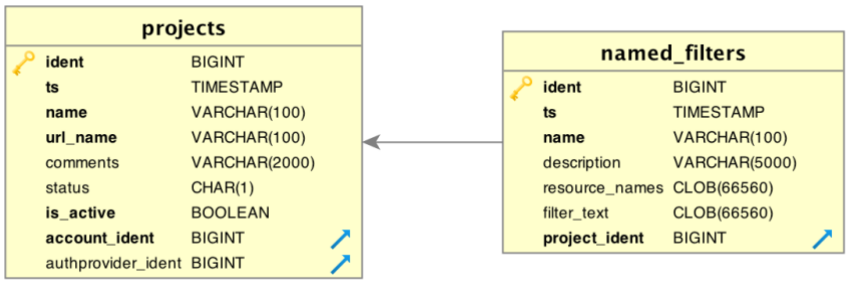
If you do not specific a file name (--file functions.json) lacadmin will output to standard out console. Note that the function returns the parameter definitions if used.

```
$lacadmin function export
[
 {
  "name": "gcd",
  "comments": "Simple function that takes two parameters…",
  "code": "// Trivial example:",
  "resource_names": null,
  "return_type": "{\n  \"n1\":6,\n  \"n2\":9,\n  \"gcd\":3\n}",
  "is_active": true,
  "project_ident": 2000,
  "parameters": [
   {
    "name": "n1",
    "param_order": 0,
    "comments": "First number",
    "is_required": true,
    "type_ident": 1
   },
   {
    "name": "n2",
    "param_order": 1,
    "comments": "Second number",
    "is_required": true,
    "type_ident": 1
   }
  ]
 }
]
```

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:functions

Filters (userfilter)

The named filter is used to define the arguments and syntax used to query SQL databases. The filter text is passed as part of the where clause and can replace {argnames}, thus masking the actual attribute name. See Structured Filters documentation.




```
$lacadmin filter --help
```

Usage: filter [options] <list|create|delete|update|import|export>

Administer Named filter for the active API Project.

Options:

-h, --help	output usage information
--ident [ident]	The ident of the specific named filter object
--filtername [name]	The Name of named filter
--filter_text [text]	Text to define named filter
--resource_names [name]	[Optional] Comma seperated list of Resource Names in quotes
--comments [comment]	[Optional] omment on named filter
--project_ident [project_ident]	[Optional] The project ident if not the active project
--file [fileName]	[Optional] Name of file for import/export (if not provided stdin/stdout used for export)
--verbose	[Optional] whether to display list of named filter in detailed format

Export/Import

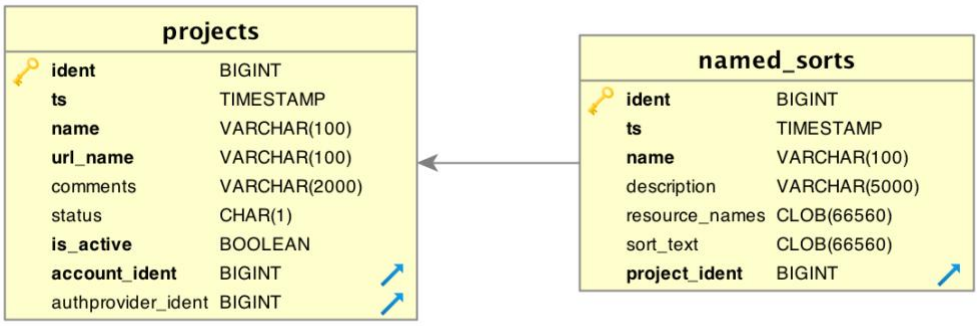
```
$lacadmin filter export
[
 {
```

```
    "ts": "2016-12-29T02:11:34.436",
    "name": "NamedFilterSample",
    "description": "Sample Description",
    "resource_names": null,
    "filter_text": "balance  < {balance}",
    "project_ident": null
  }
]
```


REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:named_filters

Sorts (usersort)

Named Sorts are user defined SQL Order By clause with optional {argname}. See Structured Sorts documentation.



$lacadmin sort --help

Usage: sort [options] <list|create|update|delete|import|export>

Administer Named Sorts for the active API Project.

Options:

-h, --help output usage information
--ident [ident] The ident of the specific named sort object
--sortname [name] The Name of named sort
--sort_text [sorttext] Sort Text to define named sort
--resource_names [name] [Optional] Comma seperated list of Resource Names in quotes

```
--comments [comment]            [Optional] Comment on named sort
--project_ident [project_ident]  [Optional] The project ident if not the active project
--file [fileName]               [Optional] Name of file for import/export (if not
                                           provided stdin/stdout used for export)
--verbose                       [Optional]  whether to display list of named sorts in
                                           detailed format
```

Export/Import

```
$lacadmin sort export
[
 {
   "ts": "2016-12-29T02:13:23.618",
   "name": "SampleSort",
   "description": "Sort Description",
   "resource_names": null,
   "sort_text": "name desc",
   "project_ident": null
 } ]
```

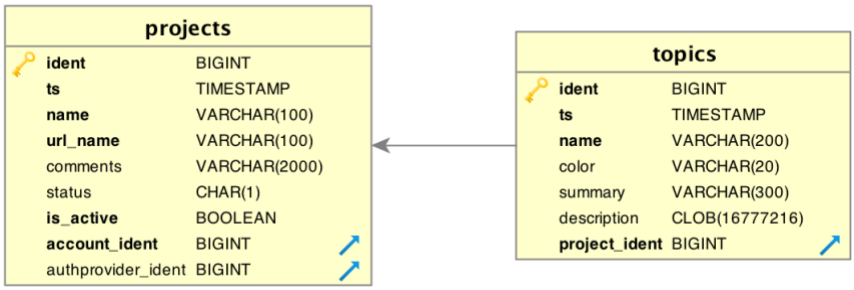
Update

The update command will allow changes to the --sort_text using the--ident for a specific row.

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:named_sorts

Topics

Topics are documentation entries and are associated with one or more reactive logic rules. This gives users the ability to group rules by topic and add documentation detail (and color coded) descriptions to groups of rules. See Manage Topics documentation.


projects
ident BIGINT
ts TIMESTAMP
name VARCHAR(100)
url_name VARCHAR(100)
comments VARCHAR(2000)
status CHAR(1)
is_active BOOLEAN
account_ident BIGINT
authprovider_ident BIGINT
topics
ident BIGINT
ts TIMESTAMP
name VARCHAR(200)
color VARCHAR(20)
summary VARCHAR(300)
description CLOB(16777216)
project_ident BIGINT

```
$lacadmin topic --help
```

```
Usage: topic [options] <list|delete|export|import>

Administer Topics for current project.

Options:

  -h, --help                    output usage information
  --project_ident [project_ident]  The project ident that will be marked as used
  --ident [ident]                  The ident for a given topic
  --file [fileName]             [Optional] Name of file to import/export
```

List

```
$lacadmin topic list
```

```
Topic
Ident  Name                Color    Summary                                                                  Description
-----  ------------------  -------  -----------------------------------------------------------------------  ---------------------------------------------------
2016   Check Credit          #98fb3a  Balance < CreditLimit -- 5 rules vs hundreds of lines of code          <p>5 rules == several hundred lines of code, an...
2018   Healthy Food Discount  #00fcfb  Discounts for eating healthy (fruit or fish) (more than 3 such items)  <p>The logic defined for this topic illustrates...
2019   Salary Event Source    #ffb7ef  Illustrate Event Source pattern for giving raise                       <h2>Business Requirement</h2>  <p>When a row is...
2020   Shipper Alert          #fff7a9  Alert Ship-to of any changes to order                                  <h2>Business Background</h2>  <p> </p>

<h...
2021   Supplier Alert         #ff899b  Alert Supplier (e.g., Pavlov) per agreed-upon API.                    <h2>Business Background</h2>  <p> </p>

<h...
2017   eMail Alerts           #8abafb  Alert the Sales Manager, SalesRep.                                    <p>Illustrates</p>  <ul>
              <li>Loadable Librarie...

# topics: 6
```

Export/Import

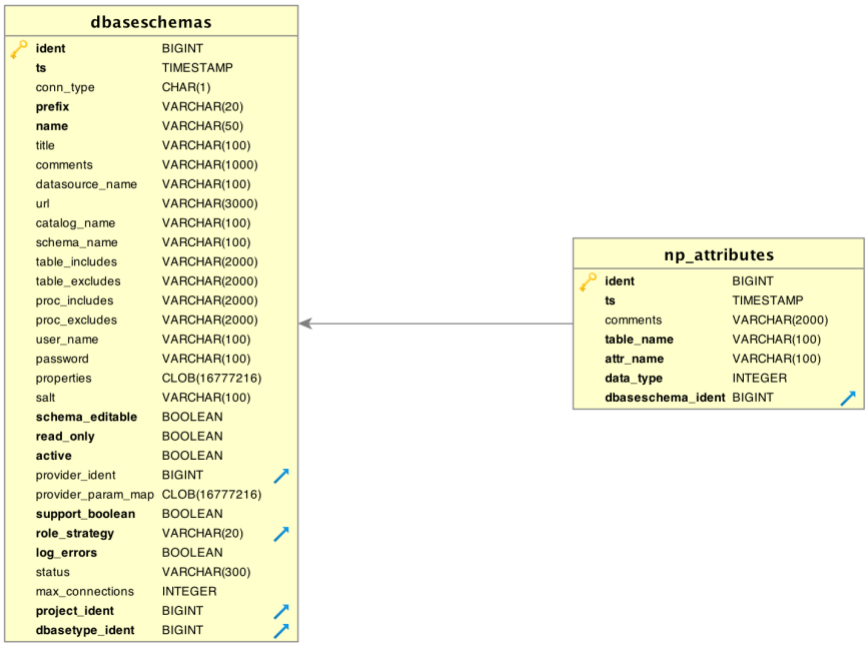
```
$lacadmin topic export
[
 {
   "ts": "2016-12-29T20:53:45+00:00",
   "name": "Audit Orders",
   "color": "#00fcee",
   "summary": "If amount changes",
```

"description": " This examples illustrates that your logic is a combination of Reactive Programming Rules, and JavaScript. \n\n Note your JavaScript can use LIbraries of Java/JavaScript code. Upload these to your project in Project properties. \n\n   \n"
}
]

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:topics

Non-Persistent Attributes (npa)

This feature allows virtual attributes to be added to tables, however, these attributes will not be persisted to the database. Non-persistent attributes may be used in rules like sum, count, min, max, and formula and appear as new columns in schema, normal resources, and data explorer. See Non-Persistent Attributes documentation.



$lacadmin npa --help

Usage: npa [options] <list|delete|create|export|import>

Administer Non Persistent Attributes for the active API Project.

Options:

-h, --help output usage information

--ident [ident] The ident of the specific named sort object

--dbschema_ident [ident] [Optional] The dbschema ident if not the active project

--file [fileName] [Optional] Name of file for import/export (if not provided stdin/stdout used for export)

--verbose [Optional] whether to display list of named sorts in detailed format

List

```
$lacadmin npa list
```

```
Non Persistent Attributes Schema Name: Derby - Northwind
Ident  Table Name     Attr Name         Data Type  Comments
-----  -------------  ----------------  ---------  --------
2003   Employees      fullName          Character
2000   Order Details  isHealthy         Boolean
2002   Orders         discountedAmount  Decimal
2001   Orders         healthyCount      Integer
2004   Suppliers      URL               String

# non persistent attrs: 5
```

Export/Import

```
$lacadmin npa export
[
 {
   "ts": "2016-12-29T02:16:04.38",
   "comments": "sample npa count",
   "table_name": "customer",
   "attr_name": "npaCount",
   "data_type": 4,
   "dbaseschema_ident": 2000,
   "prefix": "demo"
 }
]
```

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:np_attributes

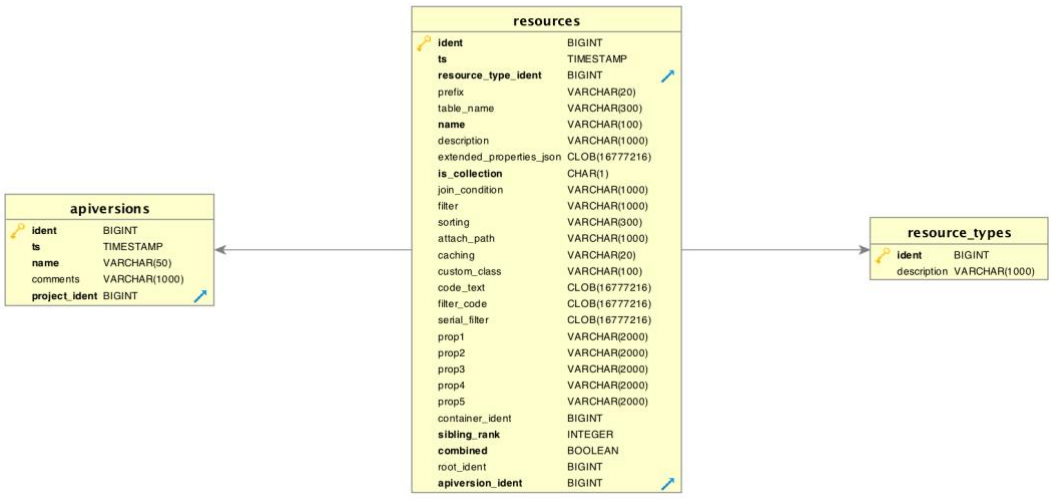
Resources

There are currently 4 types of resources (Normal (entity-based), JavaScript, FreeSQL, and MongoDB). Normal resources are based on entities (tables or views) can have multiple nested

levels and attributes may have alias names (see resource attributes). FreeSQL and JavaScript use the various fields to store the definitions (prop[n]). The new api section format is recommended for source control.

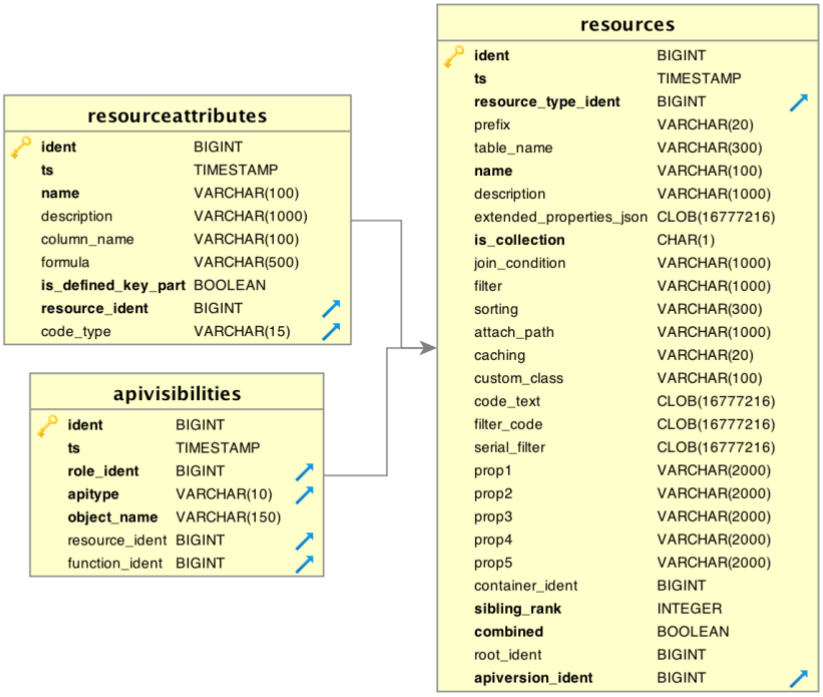
$lacadmin api export –-section resources –-section_filter “version=v1,name=AllCustomers” –-format json

See Customize your API documentation.



While it is possible to create a resource via REST (which is how it is done in the IDE) – the lacadmin allows for export and import of existing defined resources of any type. Given the ability for resources to be ‘nested’ – the code to process or update a resource is recursive. In other words, it knows how to put the children back with the proper parents (note the self relationships above) using the exported ident columns.

Resource Attributes



`$lacadmin resource --help`

Usage: resource [options] <list|delete|update|export|import>

Administer resources within a project.

Options:

-h, --help output usage information
--ident [ident] For update, the ident of the resource (use resource list)
--prop1 [value] For update, the server name of the mongo resource
--prop2 [value] For update, the database name of the mongo resource
--prop3 [value] For update, the user name of the mongo resource
--prop4 [value] For update, the password name of the mongo resource
--table_name [tablename] For update, the table name of the normal or mongo resource
--resource_name [resourcename] The name of the resource

--type [type] The type of the resource: normal, sql, javascript, storedproc, mongo

--prefix [prefix] The prefix of the table

--apiversion [apiversion] The name of an API version, if there is more than one – default v1

--project_ident [ident] The ident of a project, (if other than the current project

--file [file] Optional: for import/export, the name of a file to read

List

```
$lacadmin resource list
```

```
Top-level resources for API Version: 2003
Ident  Name               Prefix  Entity     Type    Comments
-----  -----------------  ------  ---------  ------  ----------------------------------------------------
2045   EmployeesWithRaises  nw      Employees  normal
2051   OrderSummary         nw      Orders     normal  Used by Partners to create an order, per our AP...
2052   PartnerOrder         nw      Orders     normal  Create order by posting this from Partner,with ...
2053   ShipperAPIDef        nw      Orders     normal  Used by Orders.ShipperAlert to transform Orders...
2054   SupplierAlert        nw      Orders     normal  Post to Supplier (e.g., Pavlov),with attribute ...
2063   SupplierInfo         nw      Suppliers  normal  Integrates multiple databases, joining data fro...
2064   SuppliersObject      nw      Suppliers  normal

# resources: 7
```

Export/Import

Resources can be exported one at a time or in bulk (remove the --ident) and the nested resources appear at the same level but are linked using the container_ident. The Live API Creator GUI also supports individual resource export/import.

```
$lacadmin resource export --ident 2000
[
 {
   "ident": 2000,
   "resource_type_ident": 1,
   "entity_name": "demo:customer",
   "prefix": "demo",
   "table_name": "customer",
   "name": "AllCustomers",
   "description": "Query for all customers",
   "is_collection": "Y",
```

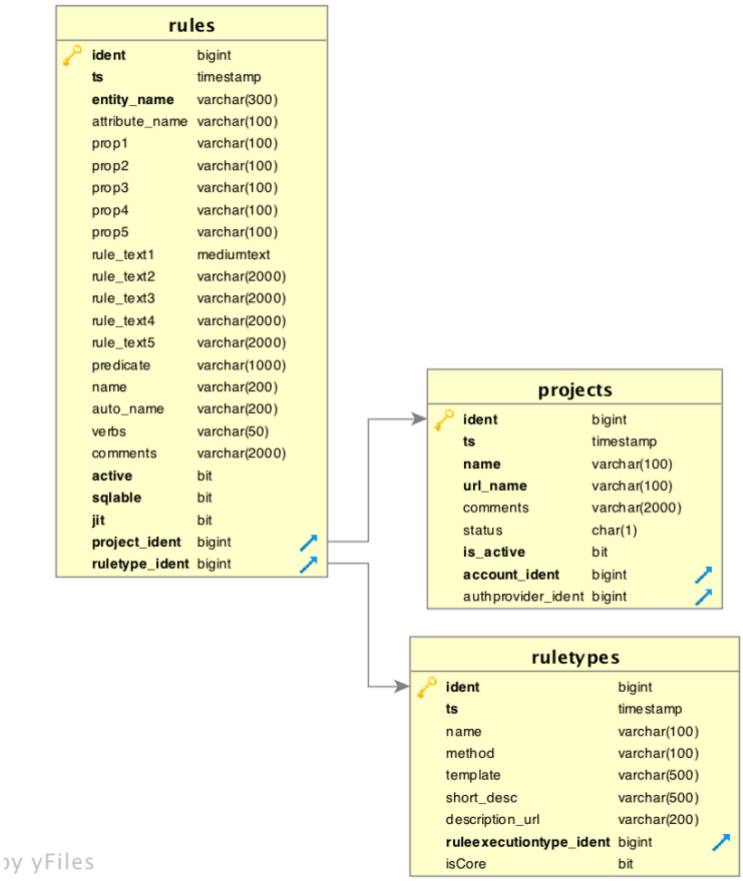
```
"join_condition": null,
"filter": null,
"sorting": null,
"caching": null,
"custom_class": null,
"code_text": null,
"filter_code": null,
"prop1": null,
"prop2": null,
"prop3": null,
"prop4": null,
"prop5": null,
"attach_path": null,
"container_ident": null,
"root_ident": null,
"sibling_rank": 100,
"combined": false,
"Attributes": [
 {
   "ident": 2001,
   "name": "balance",
   "description": "",
   "column_name": "balance",
   "format": null,
   "formula": null,
   "resource_ident": 2000,
   "is_defined_key_part": false
 },
 {
   "ident": 2000,
   "name": "name",
   "description": "",
   "column_name": "name",
   "format": null,
   "formula": null,
   "resource_ident": 2000,
   "is_defined_key_part": false
 }
]
```

```
    }
  ]
```

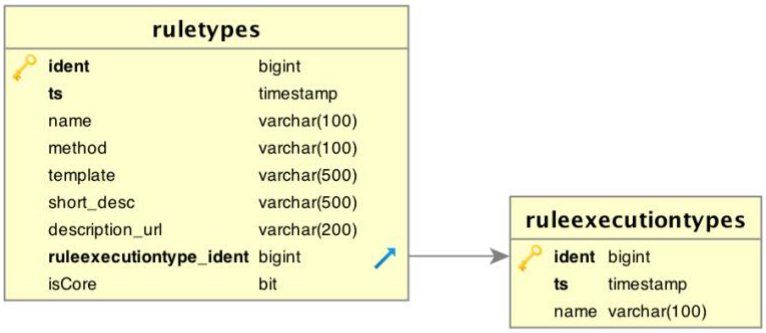
REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:resources

Rules

Reactive logic (aka Rules) can be defined on any entity (table or view). These rules are entered as derivations on column attributes (e.g. sums, counts, formula, parent copy, min, max) or events on entitles (early event, event, commit event). If you use the list command with the double dash verbose flag (--***verbose***), the output will create a view of all rules for your current project, it will also include one row for each rule in 'lacadmin rule create' format. See Rule Types documentation.



Rule Types



Using the $lacadmin **rule list–verbose** flag, rules can be displayed in 'create' format. Note: some of the flags may not be appropriate for certain rules (e.g. jit – just-in-time and sqlable). The following shows the various rule types and the lacadmin syntax:

```
$ladacmin rules list --verbose
```

Sum

```
$lacadmin rule create --ruletype sum
--entity_name sample:products
--attribute_name total_quantity_ordered
--rule_name null
--role_name lineitemsList
--child_attribute quantity_ordered
--clause 'is_ready = true'
--active true
--jit false
--sqlable false
--comments 'this is a sum rule'
```

Count

```
$lacadmin rule create --ruletype count
--entity_name sample:products
--rule_name null
--role_name components
--attribute_name count_components
--clause 'null'
--active true
--jit false
--sqlable false
--comments 'Count as existence check'
```

Formula

```
$lacadmin rule create --ruletype formula
--entity_name sample:products
--attribute_name amount
--rule_name ‘is_reorder_required’
--expression 'return row.quantity_on_hand - row.total_quantity_ordered <
row.quantity_reorder'
--active true
--jit false
--sqlable false
--comments 'Sample Formula expression on a column’
```

Parent Copy

```
$lacadmin rule create --ruletype parentcopy
--rule_name 'Copy Parent Product Price'
--entity_name demo:LineItem
--attribute_name product_price
--role_name product
--parent_attribute price
--comments 'test'
--active true
```

Validation

```
$lacadmin rule create --ruletype validation
--rule_name 'Validation Test'
--entity_name demo:customer
--expression 'return true;'
--error_message 'This is an error Message'
--active true
```

Commit Validation

```
$lacadmin rule create --ruletype commitvalidation
--rule_name 'Commit Validation Test'
--entity_name demo:customer
--expression 'return true;'
--error_message 'This is a commit validation (if false) error Message'
--active true
```

Event

```
$lacadmin rule create --ruletype event
```

```
--entity_name sample:products
--rule_name Controlled Substance Reporting
--expression 'if (row.name == "Dynamite" && row.total_quantity_ordered !=
oldRow.total_quantity_ordered) {
    log.debug("***controlled substance - notify authorities");
    var json = JSON.parse(req.json);  // convert req's JSON string to
JSON objects
    var custName = json.customer;     // get the name from the request
    var options = { sysfilter: "equal(name:'" + custName + "')" };
    var custAccount = SysUtility.getResource("ComplianceReporting",
options);  // get resource JSON
    log.debug("***sending compliance message" +
JSON.stringify(custAccount[0]));
}'
--active true
--comments 'ControlledSubstanceReporting - report accounts ordering
Dynamite; MakeOrderReady'
```

Pre-insert Event

```
$lacadmin rule create --ruletype event
--rule_name 'Pre-insert Event'
--entity_name demo:customer
--expression 'log.debug("My JavaScript here");'
--active true
```

Early Event

```
$lacadmin rule create --ruletype event
--rule_name 'Event'
--entity_name demo:customer
--expression 'log.debug("My JavaScript here");'
--active true
```

Commit Event

```
$lacadmin rule create --ruletype commitevent
--rule_name Commit Event Test'
--entity_name demo:customer
--expression 'return true;'
--error_message 'This is a commit event error Message (if false)'
--active true
```

Minimum/Maximum

```
$lacadmin rule create --ruletype [min|max]
--rule_name 'Max/Min'
```

```
--entity_name sample:orders
--attribute_name max_order
--role_name customer
```

Managed Parent

```
$lacadmin rule create --ruletype managedparent
--rule_name 'Managed Parent'
--role_name customer
```

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:rules

Rule Versions

This is an internal table that is used to keep prior change logs of changes using the Live API Creator IDE. Each change is managed under a new version ident so prior changes can be retrieved by timestamp (ts). Note – versions are not persisted between server restart, since you can use your source control system to check-in/check-out repository changes.

```
$lacadmin rule --help
```

```
Usage: rule [options] <list|create|delete|import|export>

Administer rules within a project.

Options:

  -h, --help                output usage information
  --ruletype [type]         The type of the rule, can be:
                                                sum,formula,validation,parentcopy
  --entity_name [prefix:table]  The table, qualified with a prefix, for the rule
  --attribute_name [name]     The name of the attribute whose value is computed by the
                                                rule. Required for sum, count, formula, minimum, maximum.
  --role_name [name]          The role name - required for sum, count, minimum, maximum
  --clause [clause]           The clause - required for sum, count, minimum, maximum
  --child_attribute [name]    The name of the child attribute - required for sum,
                                                minimum, maximum
  --parent_attribute [name]   The name of the parent attribute - required for parent copy
  --expression [code]         The code for the rule - required for formula, events and
                                                validations
```

--error_message [message] The error mesaage for the rule - required for validations

--rule_name [name] Optional: a name for the rule. If not specified, a name will be generated.

--comments [comments] Optional: a comment for the rule

--active [true|false] Optional: whether the rule should be active, true by default

--project_ident [ident] The ident of a project, if other than the current project

--ident [ident] For delete, the ident of the rule to delete

--jit [true|false] Just in time flag (default false)

--sqlable [true|false] Sqlable flag (default false) - optimize using SQL instead of JavaScript (default false)

--file [file] Optional: for import/export, the name of a file to read from/save to, if unspecified, use stdin/stdout

--verbose Optional: whether to display list of rules in detailed format that can be used to recreate using command line

List

If you add the double-dash verbose flag (--verbose) to the list command – it will print each rule out in a full create syntax for that rule (shown above).

```
$lacadmin rule list
Rules
Ident  Table            Type  Description                                  Comments
-----  ---------------  ----  -------------------------------------------  ------------------------------------------------------------
2041   nw:Customers           Derive Balance as sum(OrdersList.discountedAmount wher...  adjust the balance to be the sum(OrdersList.AmountTota...
2042   nw:Customers           Validation return row.Balance <= row.CreditLimit;          Observe Error message insertion points {}
2043   nw:EmployeeRaises      Event: B2B.copyAttributes(logicContext, row.Employe...     On insert, this copies the row.Salary to the Employ...
2044   nw:Employees           Derive fullName as return row.TitleOfCourtesy + " "...
2045   nw:Order Details       Derive Amount as var amount = row.Quantity * row.Un...     JavaScript is used to express logic,providing acces...
2047   nw:Order Details       Derive UnitPrice as parentcopy(FK_Order_Details_Pro...      Obtain the price from the product.Copy means subseq...
2046   nw:Order Details       Derive isHealthy as // logicContext.logDebug("compu...
2048   nw:Order Details       Event: var product = row.FK_Order_Details_Products;...     if orderDetails' products' supplier has a WebHook U...
2054   nw:Orders              Commit event: var mail = B2B.sendEmail();     // th...
2055   nw:Orders              Commit event: var salesRep = row.EmployeeID;      /...
2053   nw:Orders              Commit event: var shipper = row.FK_Orders_Shippers;...     If this order shipped by an external Shipper, alert...
2052   nw:Orders              Commit event: var webHooks = req.getUserProperties(...     for each WebHook accrued in Order Items, post -- B2...
2049   nw:Orders              Derive AmountTotal as sum(Order_DetailsList.Amount)        Adjust the AmountTotal to be the sum(Order_DetailsL...
2050   nw:Orders              Derive discountedAmount as // log.debug("isHealtyCo...
2051   nw:Orders              Derive healthyCount as count(Order_DetailsList wher...
2056   nw:Suppliers           Derive URL as return req.localFullBaseURL.replace("...     normally entered, here just computed to illustrate ...

# rules: 16
```

Export/Import

While each rule uses different attributes – the structure of both the database and JSON is the same.

```
$lacadmin rule export --ident 2003
[
 {
   "ts": "2016-12-03T06:13:24.482",
   "entity_name": "demo:LineItem",
   "attribute_name": "amount",
   "prop1": null,
   "prop2": null,
   "prop3": null,
   "prop4": "javascript",
   "prop5": null,
   "rule_text1": "if (row.qty_ordered <= 6)  // discount (using conditional JavaScript logic)\n   return row.product_price *
row.qty_ordered;\nelse\n   return row.product_price * row.qty_ordered * 0.8;\n",
   "rule_text2": null,
   "rule_text3": null,
   "rule_text4": null,
   "rule_text5": null,
   "predicate": null,
   "name": "Discounted price*qty",
   "auto_name": "Derive amount as if (row.qty_ordered <= 6)  // discount (using conditional JavaScript logic)\n   return
row.product_price * row.qty_ordered;\nelse\n  ...",
   "verbs": null,
   "comments": "Reactive Logic is expressed in JavaScript, so you use...\n- conditional logic (as above),\n- standard
JavaScript services (e.g., moment date functions - enable in Project > Libraries),\n- SQL / external services....
whatever is required.",
   "active": true,
   "sqlable": false,
   "jit": false,
   "ruletype_ident": 3,
   "@metadata": {
     "href": "http://localhost:8080/rest/abl/admin/v2/admin:rules/2003",
     "checksum": "A:fdd6caf45b3afd7c"
   }
 }
]
```

Virtual Primary Keys (Tables/Views)

CA Live API Creator provides the ability to define virtual primary keys on both tables and views. A virtual key on a view will allow create an href link in the @metadata during a GET. It will also be the first step in making a view updateable. See documentation on updates on views.


view_infos
ident bigint
ts timestamp
view_name varchar(500)
primary_key_definition varchar(2000)
sequence_info varchar(1000)
autonum_columns varchar(500)
dbaseschema_ident bigint
dbaseschemas
ident bigint
ts timestamp
conn_type char(1)
prefix varchar(20)
name varchar(100)
comments varchar(1000)
datasource_name varchar(100)
url varchar(3000)
catalog_name varchar(100)
schema_name varchar(100)
table_includes varchar(2000)
table_excludes varchar(2000)
proc_includes varchar(2000)
proc_excludes varchar(2000)
user_name varchar(100)
password varchar(100)
salt varchar(100)
schema_editable bit
read_only bit
active bit
role_strategy varchar(20)
log_errors bit
status varchar(300)
project_ident bigint
dbasetype_ident bigint


table_infos
ident bigint
ts timestamp
table_name varchar(100)
sequence_info varchar(1000)
primary_key_definition varchar(2000)
dbaseschema_ident bigint
dbaseschemas
ident bigint
ts timestamp
conn_type char(1)
prefix varchar(20)
name varchar(100)
comments varchar(1000)
datasource_name varchar(100)
url varchar(3000)
catalog_name varchar(100)
schema_name varchar(100)
table_includes varchar(2000)
table_excludes varchar(2000)
proc_includes varchar(2000)
proc_excludes varchar(2000)
user_name varchar(100)
password varchar(100)
salt varchar(100)
schema_editable bit
read_only bit
active bit
role_strategy varchar(20)
log_errors bit
status varchar(300)
project_ident bigint
dbasetype_ident bigint

Help

```
$lacadmin virtualkey --help

  Usage: virtualkey [options] <list|create|update|delete|import|export>

  Manage a virtualkey to a table or view.

  Options:

    -h, --help                        output usage information
    --table_ident [ident]             For delete or update, the ident of the listed
                                      table
    --view_ident [ident]              For delete or update, the ident of the listed
                                      view
    --project_ident [project_ident]   The project ident that will be used to list all
                                      datasources
    --prefix [prefix]                 The datasource prefix for this table or view
                                      virtual primary key
    --table_name [name]               The name of the table to attach a virtual primary
                                      key
    --view_name [name]                The name of the view to attach a virtual primary
                                      key
    --keyname [colnamelist]           The comma separated list of column names
    --is_autonum [true|false]         If the keyname of a view column that is an
                                      autonum - default false
    --file [fileName]                 [Optional] Name of file to import/export (if not
                                      provided stdin/stdout used for export)
```

list

```
$lacadmin virtualkey list
```

The list command shows all virtual keys for both tables and views.

Output

```
Virtual Primary Key
prefix   Active  view_ident  view_name              key name       autonums
table_ident  table_name
-------  ------  ----------  ---------------------  -------------  --------  ---------
--  --------------------
demo     true    2009        customers_owing        "name"
                 2010        employee_with_picture  "employee_id"
                 2011        LineItemJoinProduct    "LineItemId"
                 2014        v_LineItem             "LineItemId"
                 2016        v_LineItem2            foo2

demo     true                                       "login"                 2014
employee
```

```
                                                    "id"                    2015
STRESS_NO_PRIMARY_KEY
finance  true
```

create (view)

The create command creates a new virtual primary key. Note - only views need to indicate if the column is defined as an autonum (boolean).

```
$lacadmin virtualkey create --view_name v_LineItem
    --keyname LineItemId
    --prefix demo
    --is_autonum true
```

create (table)

```
$lacadmin virtualkey create --table_name v_LineItem
    --keyname LineItemId
    --prefix demo
```

update (view)

```
$lacadmin virtualkey update --view_ident 2016
--view_name v_LineItem
--keyname LineItemID
--is_autonum false
```

update (table)

```
$lacadmin virtualkey update --table_ident 2015
--table_name STRESS_NO_PRIMARY_KEY
--keyname id
```

delete

```
$lacadmin virtualkey delete [--view_ident <ident> | --table_ident <ident>]
```

The delete command deletes a specific virtual primary key for a view or table using the ident (use lacadmin virtualkey list)
Visit the Documentation page on virtualkey

export

Provide the optional prefix of the AllEntitiesInfo and (optional) the export file name. If not provided - it will be sent to stdout.

```
lacadmin virtualkey export  [--prefix <name> ] --file datasource.json
```

The export virtual primary key exports the specified definitions into a JSON file. If the filename parameter is not specified, stdout is used.

import

Import a virtual key definition to the current project (or one specified) using the name of the json file.

```
lacadmin virtuakey import [--project_ident <ident>] --file datasource.json
```

The import command will import virtual primary key from the specified JSON file. If the filename parameter is not specified, stdin is used. (you can pipe the json file to the import)

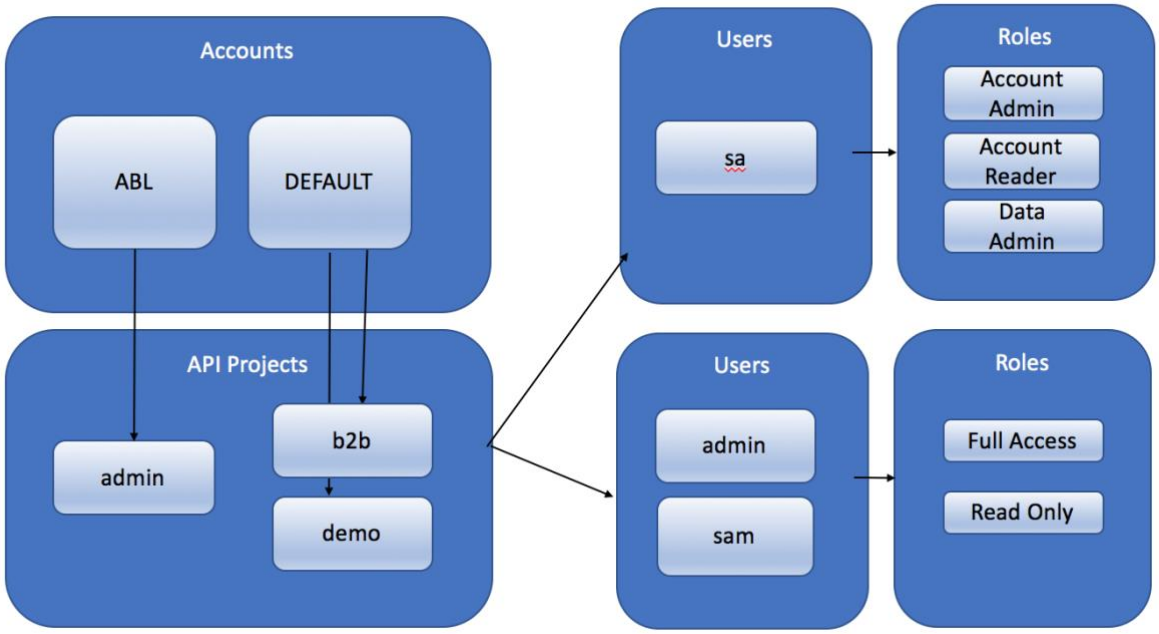
REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:view_infos

REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:table_infos

Security

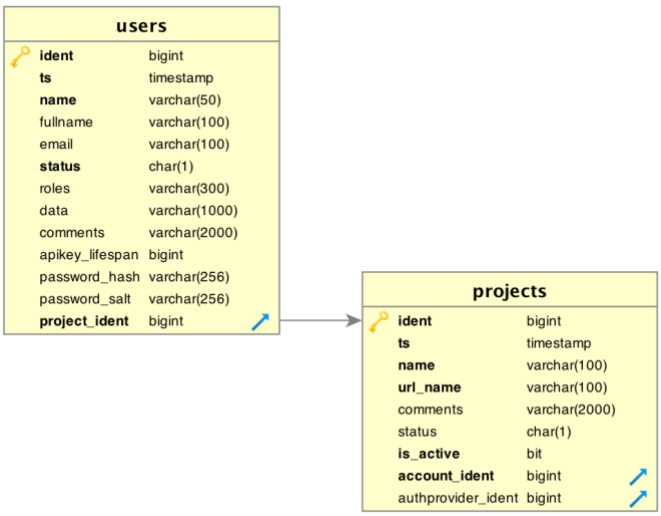
Role Based access security controls how endpoints are exposed. Once a user has been authenticated (either using the built-in service or an external service like LDAP, REST, or SQL), the authentication returns a list of roles linked to a generated auth token (aka API Key). Each subsequent REST call uses this authentication token. When you export a project, all user roles, permissions, and security settings are contained in the JSON exported file. See Security documentation. The 2 types of accounts (abl and default) are linked to projects. Each project has an authentication provider which will validate a user and the role(s) the user will play. The roles

determine access control to REST endpoints and additional security settings.



User

This only applies to the built-in authentication service. This is simply a list of user login names, roles (comma separated list), and the password.



```
$lacadmin user --help
```

```
Usage: user [options] <list|delete|update|export|import>

Administer Users for current project.

Options:

  -h, --help                    output usage information
  --project_ident [project_ident]  The project ident that will be marked as used
  --ident [ident]               The ident of the user
  --username [name]             The name of the user
  --file [fileName]             [Optional] Name of file to import/export (if not
                                                    provided stdin/stdout used for export)
  --password [password]         The password for this user
  --fullname [fullname]         User fullname
  --name [name]                 User name
  --status [status]             Status active A or inactive I
  --roles [roles]               Comma separated list of role names
  --comments [comments]         User comments
```

List

```
$lacadmin user list
```

```
Users
Ident  Name    Full Name                        Status  Roles                Life Span  Description
-----  ------  -------------------------------  ------  -------------------  ---------  -----------
2007   admin   admin user - full access           A       Full access,         86400
2008   demo    Demo Account with full access      A       Full access          null
2009   guest   Guest Account with read-only access  A       Read only            null
2010   region  User with specified region         A       Authorized per region  null
```

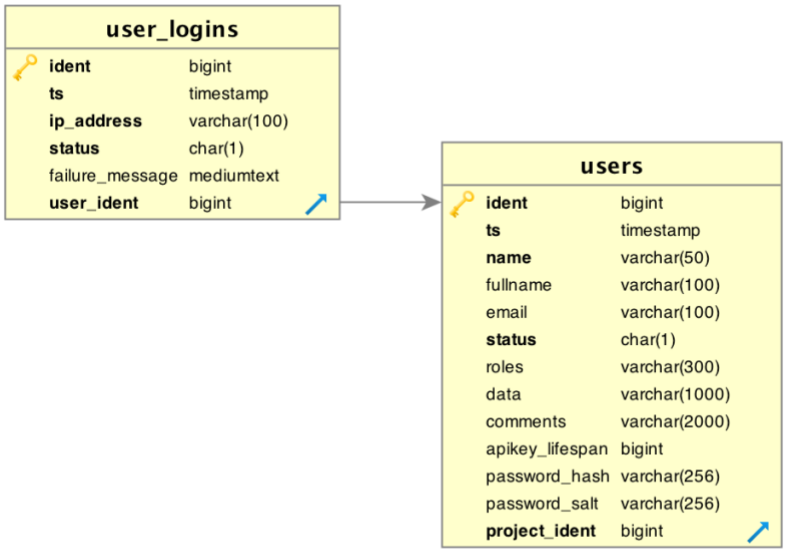
Export/Import

```
$lacadmin user export
[
 {
   "ts": "2016-12-03T06:13:26.027",
   "name": "admin",
```

```
    "fullname": "Demo user (admin)",
    "email": "admin@ca.com",
    "status": "A",
    "roles": "API Owner",
    "data": "",
    "comments": null,
    "apikey_lifespan": null
  },
  {
    "ts": "2016-12-03T06:13:26.064",
    "name": "demo",
    "fullname": "Demo user",
    "email": "admin@ca.com",
    "status": "A",
    "roles": "API Owner",
    "data": "",
    "comments": null,
    "apikey_lifespan": null
  }
]
```

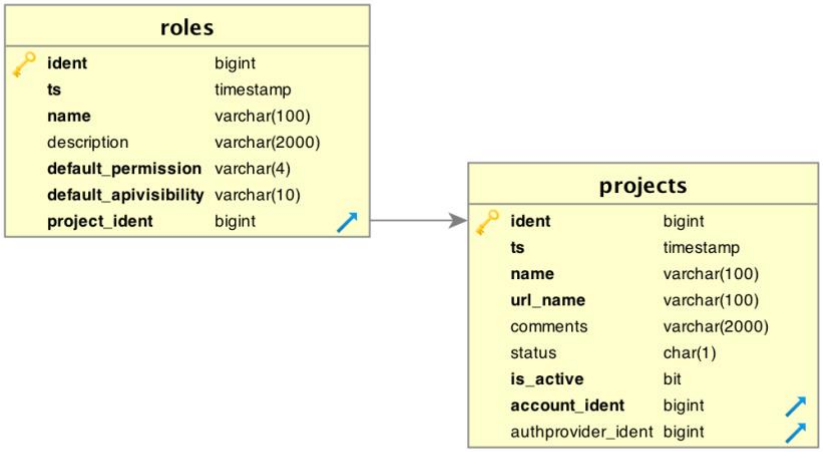

User Login

This is an internal table keeping a running list of user login requests.



Roles

A User is associated with at least one or more roles. A role defines the access control to Tables, Views, Procedures, Functions, Resources, and Meta tables (default API visibility – TVPRMF – table, view, procedure, resource, metadata, function). Additional entity permissions can be added at the table/view level as well as additional row and column filters. See Role based endpoint access documentation.



```
$lacadmin role --help

  Usage: role [options] <list|delete|export|import>

  Administer Roles for current project.

  Options:

    -h, --help                          output usage information
    --project_ident [project_ident]     The project ident that will be marked as
                                        used
    --ident [ident]                     The ident of the role
    --file [fileName]                   [Optional] Name of file to import/export
```

Default Permissions (A, N, R)

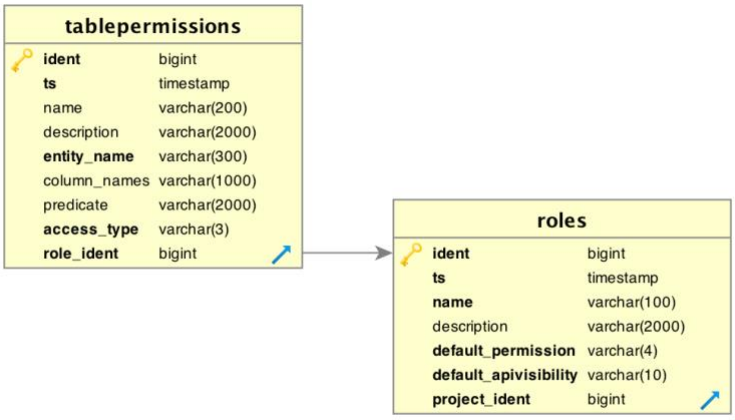
This will be one of the following: All, None, Read-only

Default Visibilities (TVPRMF)

This is a combination of these: Tables, Views, Procedures, Resources, Meta, Functions

Table Permissions

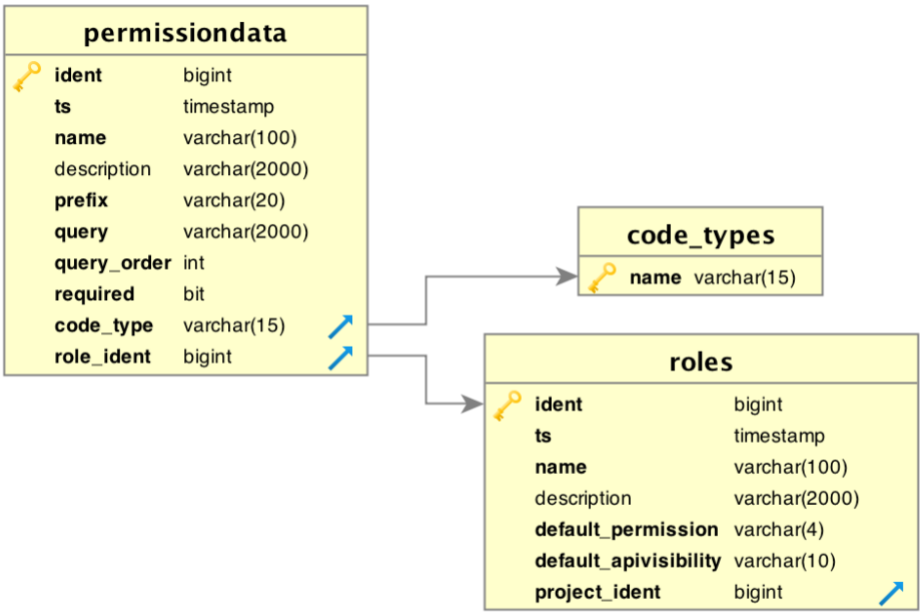
Each role may further restrict both row and column level access to table.



REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:tablepermissions

Permission Data

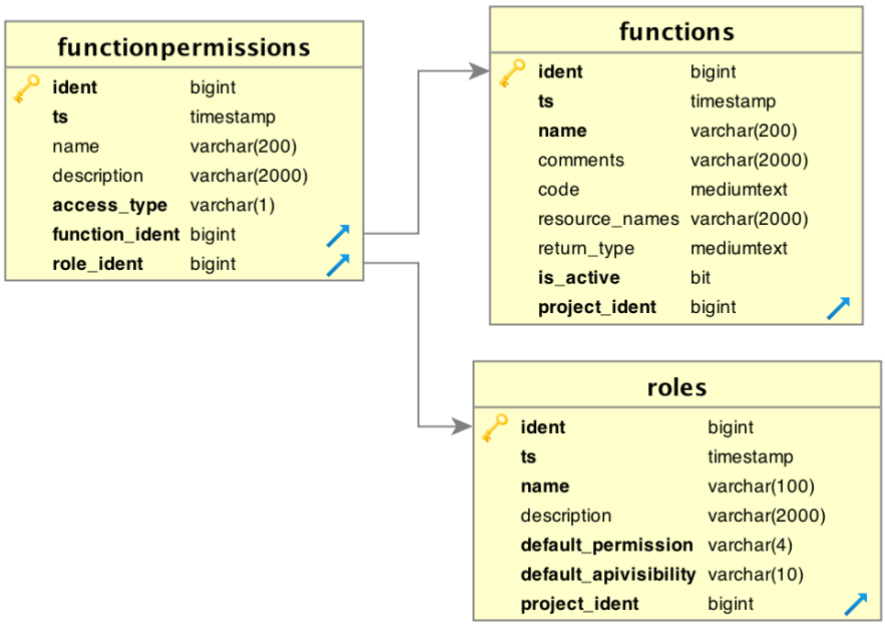
Permission data defines the entity (table/view) (name) and prefix (found on dbaseschema) with a block of query code that is injected into the where clause to determine and filter returned and updated rows.



REST Call: http://server[:port]/[war]/rest/abl/admin/v2/admin:permissiondata

Function Permissions

Function permissions determine access_type (X for execute) for one or more function endpoints. If All is granted to functions – this will be empty. This only applies for roles that require fine grain support for execute permission on function endpoints.



List

```
$lacadmin role list
```

```
Roles
Ident  Name             Visibility  Permission  Description
-----  ---------------  ----------  ----------  ------------------------------------------------
2000   API Documentation  TVRPMF      A           Full permissions on the entire API, special doc...
2001   API Owner          TVRPMF      A           Full permissions on the entire API
2002   API User           RM          N           Limited permissions on the entire API
2003   Sales Rep          RM          N           Preselects orders

# roles: 4
```

Export/Import

```
$lacadmin role export
[
 {
   "name": "API Documentation",
   "description": "Full permissions on the entire API, special documentation role",
   "default_permission": "A",
   "default_apivisibility": "TVRPMF",
```

```
  "PermissionData": [],
  "ApiVisibility": []
},
{
  "name": "API Owner",
  "description": "Full permissions on the entire API",
  "default_permission": "A",
  "default_apivisibility": "TVRPMF",
  "PermissionData": [],
  "ApiVisibility": []
},
{
  "name": "API User",
  "description": "Limited permissions on the entire API",
  "default_permission": "N",
  "default_apivisibility": "RM",
  "PermissionData": [],
  "ApiVisibility": []
},
{
  "name": "Sales Rep",
  "description": "Preselects orders",
  "default_permission": "N",
  "default_apivisibility": "RM",
  "PermissionData": [
    {
      "ident": 2000,
      "name": "current_employee_row",
      "description": null,
      "prefix": "demo",
      "query": "* from \"employee\" where \"login\" = '@{_apikey.user_identifier}'",
      "query_order": 1,
      "required": true,
      "code_type": "SQL",
      "role_ident": 2003,
      "@metadata": {
        "href": "http://localhost:8080/rest/abl/admin/v2/AllRoles.PermissionData/2000",
        "checksum": "A:c59cd0e6b519f412"
      }
```

```
      }
    ],
    "ApiVisibility": []
  }
]
```

TeamSpace Users

The teamSpace user is a list of users and system roles which control which user can logon to Live API Creator and add/edit/delete API components.

Help

```
$lacadmin teamspace_user -h

  Usage: teamspace_user [options] <list|delete|export|import>

  Administer TeamSpace Users definitions.

  Options:

    --teampspace_username [name]     The TeamSpace User Name
    --ident [ident]                  The ident of the specific TeamSpace user
    --account_ident [account_ident]  The ident of the specific TeamSpace user
    -v, --verbose                    optional: Display list of timer in
detailed export/import format
    --file [fileName]                optional: Name of file to import/export
(if not provided stdin/stdout used for export)
    -h, --help                       output usage information
```

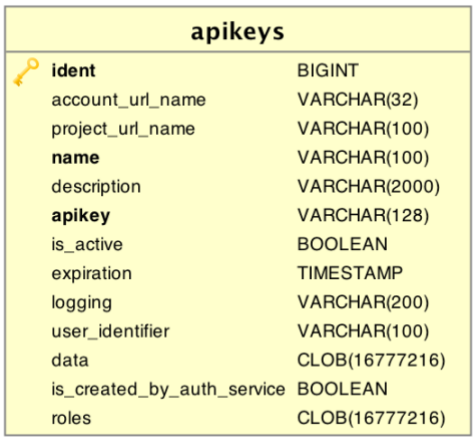
List

```
$lacadmin teamspace_user list
```

```
TeamSpace Users(s)
Ident  Name   Full Name               Status  Roles                          Comments  AccountIdent
-----  -----  ----------------------  ------  -----------------------------  --------  ------------
2001   admin  Admin user for default  A       "Account admin", "Data admin"            2000
2000   sa     System Admin            A       "System administrator"                   1
2002   tyler  Tyler user for default  A       "Account admin", "Data admin"            2001

# teamspace users(s): 3
```

API Keys (Auth Tokens)

An auth token is created for each login connection (see @authentication) (aka apikey). These keys are tied to a specific user interaction with the REST server. Auth tokens can also be created for internal use or specific roles. See Auth Tokens documentation. Note 1: Devops may wish to purge expired apikeys to clean up the database. The field data may contain additional JSON specific data provided by the auth provider which is accessible at runtime once authentication validates the user.




```
$lacadmin authtoken --help
```

Usage: token [options] <list|delete|export|import>

Administer Auth Tokens for current project.

Options:

-h, --help output usage information

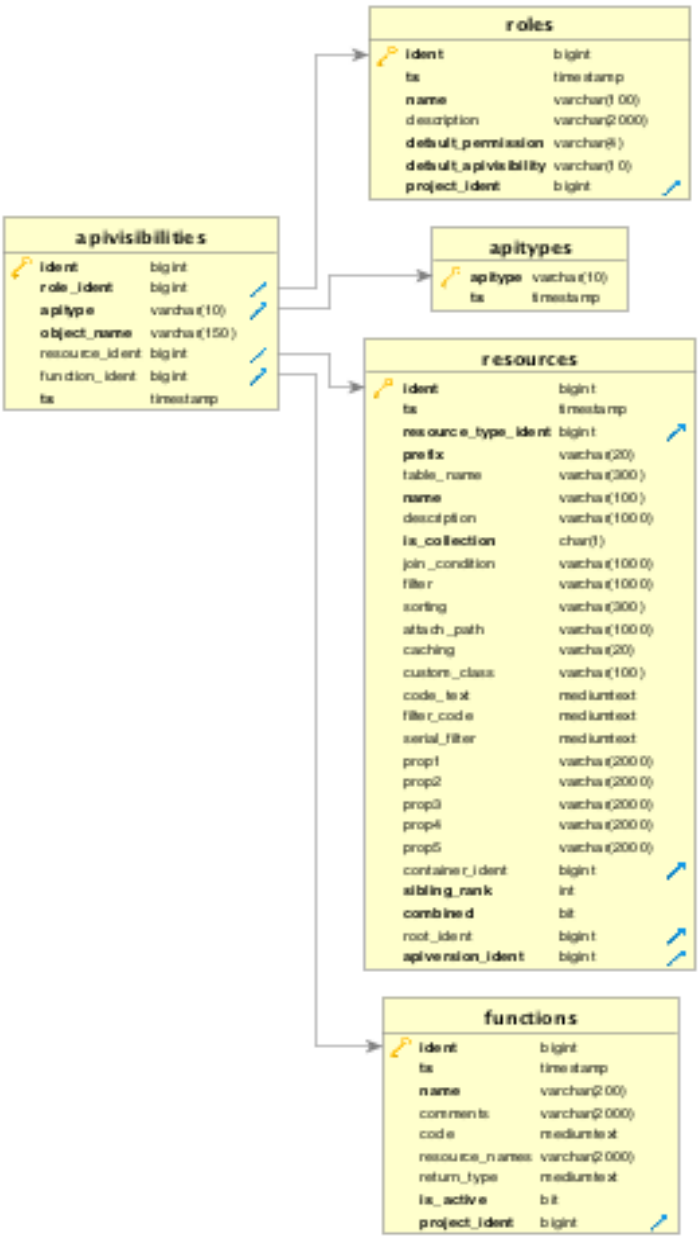
--project_ident [project_ident] The project ident that will be marked as used

--ident [ident] The ident of the token

--file [fileName] [Optional] Name of file to import/export

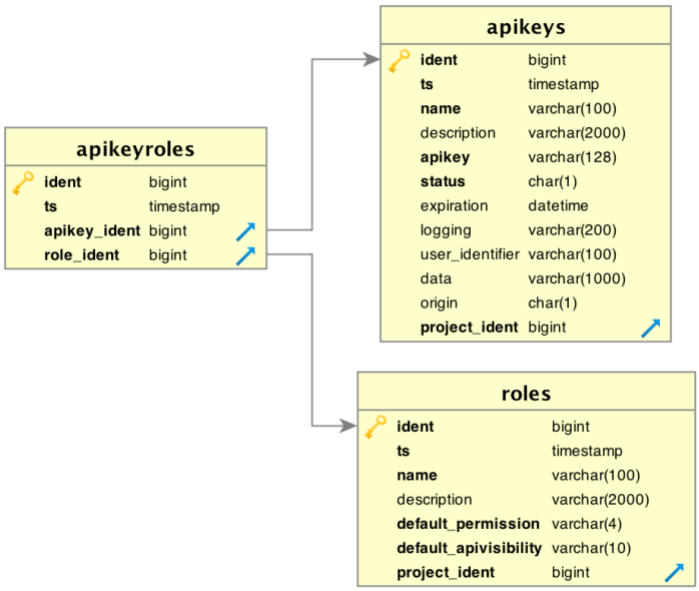
API Visibilities

This is the junction between roles and resources used to determine which specific objects are marked as enabled if the r ole visibility is not All.



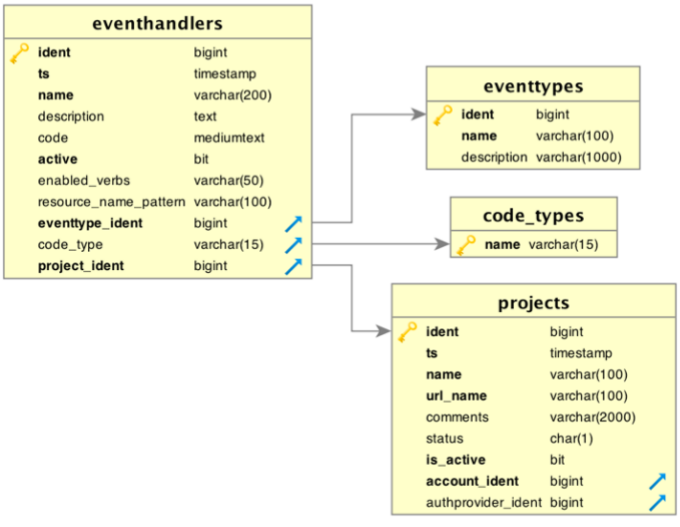
APIKey Roles

This is an internal table used to link the apikey to one or more roles.



API Event Handlers (Request/Response Events)

When a REST request is made to an endpoint, a request JavaScript event is called. When all the rule processing has completed, a response JavaScript event is called. This can be a handy way to take a GET request and do a POST to another service or modify the JSON response before returning to the caller. See Event Handlers documentation.



```
$lacadmin request_event --help
```

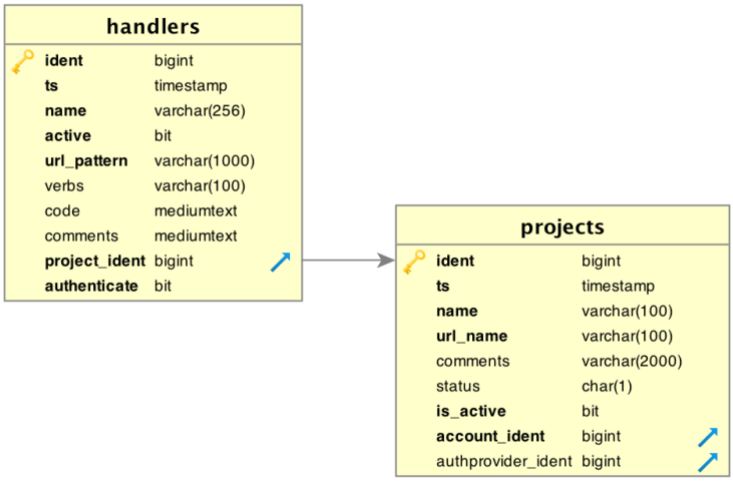
```
Usage: request_event [options] <list|delete|export|import>

Administer Request & Response Events for current project.

Options:

  -h, --help                    output usage information
  --eventname [name]              The request or response Name
  --ident [ident]                 The event ident
  --project_ident [project_ident]  The project ident that will be used
  --file [fileName]               [Optional] Name of file to import/expor
```

Export/Import

```
$lacadmin request_event export
[
 {
   "ts": "2016-12-29T02:28:57.56",
   "name": "Request Event",
   "description": null,
   "code": "// Event handling code goes here\nlog.debug(\"Request Evenrt\");",
   "active": true,
   "enabled_verbs": null,
   "resource_name_pattern": null,
   "eventtype_ident": 1,
   "code_type": "Javascript"
 }
]
```

Handlers (aka Custom Endpoints)

These are custom endpoints which can return results (like functions) but can also be authenticated or un-authenticated. An example would be to return some HTML documentation or wrapper some REST endpoint with an apikey so the REST request does not require authentication for the caller. See Custom Endpoints documentation.




```
$lacadmin handler --help
```

Usage: handler [options] <list|delete|export|import>

Administer Custom Endpoints (Handlers) for current project.

Options:

-h, --help	output usage information
--project_ident [project_ident]	The project ident that will be used
--ident [ident]	The ident for a handler
--file [fileName]	[Optional] Name of file to import/export

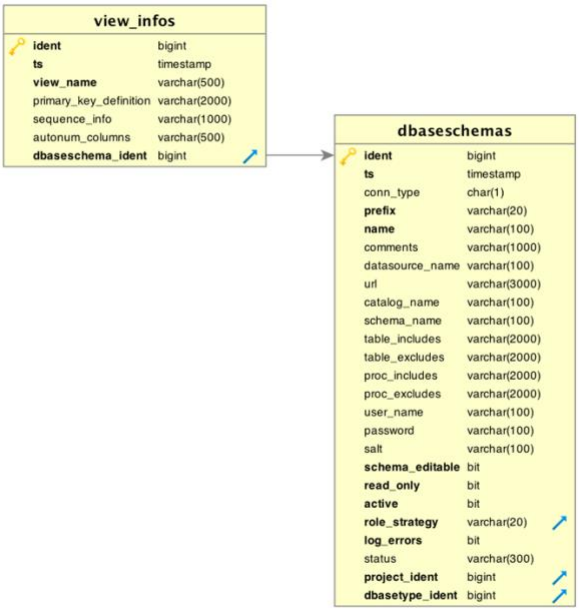
Export/Import

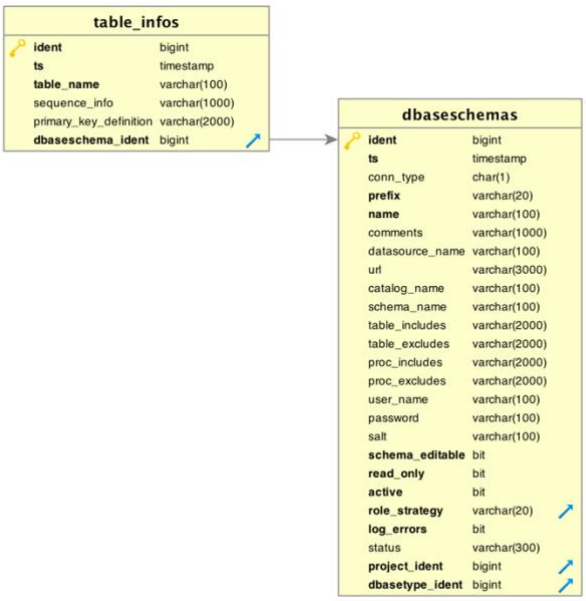
```
$lacadmin handler export
[
 {
   "ts": "2016-12-29T02:30:44.88",
   "name": "CustomEndpoint",
   "active": true,
   "url_pattern": ".*",
   "verbs": "GET,",
   "code": "var res = {result: 'Hello'};\nreturn JSON.stringify(res);\n",
   "comments": null,
   "authenticate": false
```

```
    }
]
```

Virtual Primary Keys

A virtual primary key can be defined on a table or view. The addition of a virtual primary key will be used to create the metadata href to uniquely identify each result row. If defined on a table, existing primary key will become a candidate key. If defined on a view (there can only be one) you must optionally identify it the column name is an autonum (Live API Creator knows the datatype of the attribute but on a view – it does not know the underlying database identity.





Help

$lacadmin virtualkey --help

Usage: virtualkey [options] <list|create|update|delete|import|export>

Manage a virtualkey to a table or view.

Options:

-h, --help output usage information
--table_ident [ident] For delete or update, the ident of the listed table
--view_ident [ident] For delete or update, the ident of the listed view
--project_ident [project_ident] The project ident that will be used to list all datasources
--prefix [prefix] The datasource prefix for this table or view virtual primary key
--table_name [name] The name of the table to attach a virtual primary key
--view_name [name] The name of the view to attach a virtual primary key
--keyname [colnamelist] The comma separated list of column names
--is_autonum [true|false] If the keyname of a view column that is an autonum – default false
--file [fileName] [Optional] Name of file to import/export

List

$lacadmin virtualkey list

Virtual Primary Key

prefix Active view_ident view_name key name autonums table_ident table_name

------- ------ ---------- -------------------- ------------ -------- ----------- --------------------

demo true 2057 customers_owing "name"

2058 employee_with_picture "employee_id"

2059 LineItemJoinProduct "LineItemId"

2085 v_LineItem "LineItemId"

demo true "login" 2056 employee

"id" 2057 STRESS_NO_PRIMARY_KEY

finance true

view_infos: 2

Create Virtual Primary Key (View)

The `create` command creates a new virtual primary key. Note - only views need to indicate if the column is defined as an autonum (boolean).

```
    lacadmin virtualkey create --view_name v_LineItem
    --keyname LineItemId
    --prefix demo
    --is_autonum true
```

Create Virtual Primary Key (Table)

```
    lacadmin virtualkey create --table_name v_LineItem
    --keyname LineItemId
    --prefix demo
```

Export

Shows both table and view info (AllEntityInfos resource).

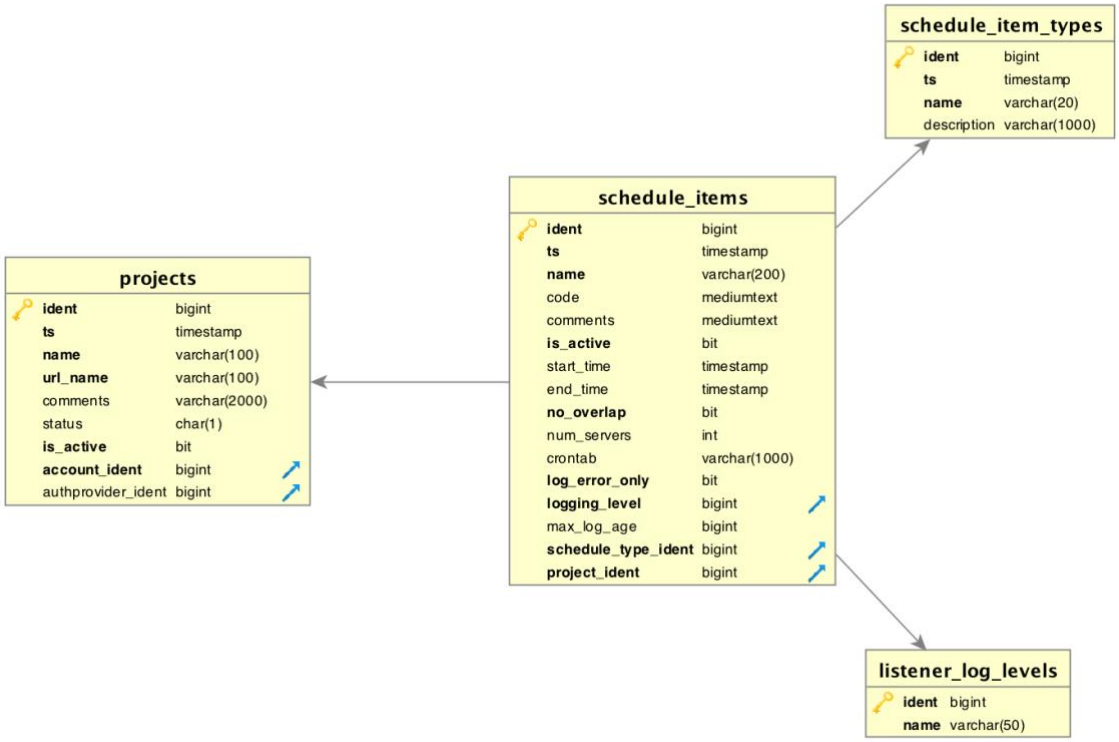
```
$lacadmin virtualkey export
[
 {
   "prefix": "demo",
   "project_ident": null,
   "active": true,
   "TableInfos": [
    {
      "ident": 2056,
      "table_name": "employee",
      "primary_key_definition": "\"login\"",
      "sequence_info": null,
      "@metadata": {
        "href": "http://localhost:8080/APIServer/rest/abl/admin/v2/AllEntityInfos.TableInfos/2056",
        "checksum": "A:5b93c844f42797ce"
      }
    }
   ],
   "ViewInfos": [
    {
      "ident": 2057,
      "view_name": "customers_owing",
      "primary_key_definition": "\"name\"",
```

```
  "sequence_info": null,
  "autonum_columns": null,
  "@metadata": {
    "href": "http://localhost:8080/APIServer/rest/abl/admin/v2/AllEntityInfos.ViewInfos/2057",
    "checksum": "A:b78f09b59b30f569"
  }
},
{
  "ident": 2058,
  "view_name": "employee_with_picture",
  "primary_key_definition": "\"employee_id\"",
  "sequence_info": null,
  "autonum_columns": "\"employee_id\"",
  "@metadata": {
    "href": "http://localhost:8080/APIServer/rest/abl/admin/v2/AllEntityInfos.ViewInfos/2058",
    "checksum": "A:4e0aad98b1d3d497"
  }
},
{
  "ident": 2059,
  "view_name": "LineItemJoinProduct",
  "primary_key_definition": "\"LineItemId\"",
  "sequence_info": null,
  "autonum_columns": "\"LineItemId\"",
  "@metadata": {
    "href": "http://localhost:8080/APIServer/rest/abl/admin/v2/AllEntityInfos.ViewInfos/2059",
    "checksum": "A:391a7d26e5bdc157"
  }
},
{
  "ident": 2085,
  "view_name": "v_LineItem",
  "primary_key_definition": "\"LineItemId\"",
  "sequence_info": null,
  "autonum_columns": "\"LineItemId\"",
  "@metadata": {
    "href": "http://localhost:8080/APIServer/rest/abl/admin/v2/AllEntityInfos.ViewInfos/2085",
    "checksum": "A:eeae4a30cc8dd903"
  }
```

```
    }
   ]
  }
]
```

Timers

Timers are scheduled events that execute user specified JavaScript code at pre-defined scheduled intervals. A timer can be listed, exported, imported, or deleted using the lacadmin command line utility.



.

Help

$lacadmin timer –help

Usage: timer [options] <list|delete|export|import>

Administer Timer definitions.

Options:

--timer_name [name] The Timer Name

--ident [ident] The ident of the specific timer

--verbose (optional) Display list of timer in detailed export/import format

--file [fileName] (optional) Name of file to import/export (if not provided stdin/stdout used for export)

-h, --help output usage information

List

$lacadmin timer list

```
Timer

Ident  Name       Active  Start Time  End Time  # Servers  Crontab        Schedule Type  Description
-----  ---------  ------  ----------  --------  ---------  -------------  -------------  -----------
2002   New Timer  false   null        null      0          /5 * * * * ? *  Repeating

# timer: 1
```

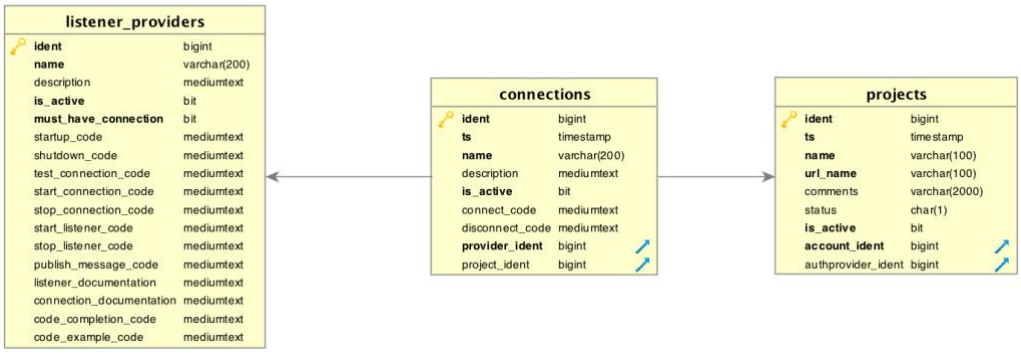
Export

$lacadmin timer export


```
[
 {
  "ts": "2017-08-14T18:19:09+00:00",
  "name": "New Timer",
  "code": "// Timer code goes here\nvar connectionName = \"MQTTConnection\";\nvar topic =\"lacresponse\";\nvar options = null;\nvar m = {\"message\": \"ALIVE!\" };\nvar success = timerUtil.publishMessage(connectionName,topic,m,options);\nif(!success) {\n    throw \"Timer failed to publish to MQTT connection \"+connectionName;\n}\nlog.debug(\"My timer has executed! a post to MQTT\");",
  "comments": "",
  "is_active": false,
  "start_time": null,
  "end_time": null,
  "no_overlap": false,
  "num_servers": 0,
  "crontab": "/5 * * * * ? *",
  "log_error_only": false,
  "logging_level": 4,
  "max_log_age": 2592000,
  "schedule_type_ident": 2
 }
]
```

Connections

The connection object defines the listener provider for a specific external system like MQTT, KafkaConsumer, KafkaProducer, JMS, or RabbitMQ. A connection may be used by one or more listeners and can be called to publish messages to these external systems.



Help

$lacadmin connection --help

Usage: connection [options] <list|delete|export|import|stop|start>

Administer Connections for current project.

Options:

--connection_name [name] The connection name
--ident [ident] The ident of the specific connection
--project_ident [project_ident] The project ident that will be used
--verbose (optional) Display list of connection in detailed export/import format
--file [fileName] (optional) Name of file to import/export (if not provided stdin/stdout used for export)
-h, --help output usage information

List

$lacadmin connection list

```
Connections
Ident  Name           Type  Active  Connect Code  Disconnect Code
-----  -------------  ----  ------  ------------  ---------------
2002   MQTTConnection  MQTT  false

# connection: 1
```

Export

$lacadmin connection export

```
[
 {
   “name”: “MQTTConnection”,
   "description": null,
   "is_active": true,
   "connect_code": null,
   "disconnect_code": null,
   "provider_ident": 3,
   "ConnectionParameters": [
    {
      "value": "tcp://localhost:1883",
      "type_ident": 1,
      "description": null
    },
    {
      "value": "RANDOM",
      "type_ident": 3,
      "description": null
    },
    {
      "value": "true",
      "type_ident": 10,
      "description": null
    },
    {
      "value": "0",
      "type_ident": 13,
      "description": null
```

```
    }
   ]
  }
]
```

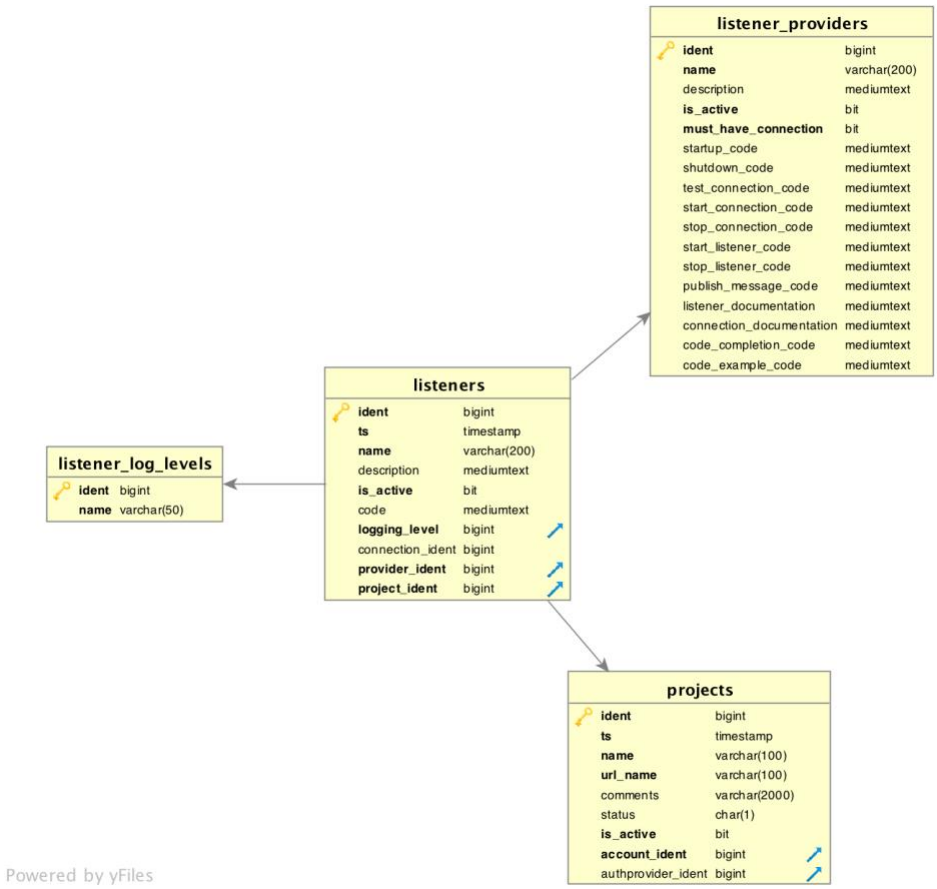
Start/Stop

This command is useful to restart a connection after import.

```
$lacadmin connection stop –connection_name MQTTConneciton

$lacadmin connection start –connection_name MQTTConneciton
```

Listeners

The listener object defines the code to handle the receipt of a specific message from an external system like Startup, Shutdown, MQTT or KafkaConsumer.



Help

$lacadmin listener --help

```
Usage: listener [options] <list|delete|export|import>

Administer Listener Events for current project.


Options:

  --listener_name [name]          The Listener Name
  --ident [ident]                 The ident of the specific listener
  --project_ident [project_ident]  The project ident that will be used
  --verbose                       (optional) Display list of listeners in detailed
                                                    export/import format
  --file [fileName]               (optional) Name of file to import/export (if not
                                                    provided stdin/stdout used for export)
  -h, --help                      output usage information
```

List

List of all listener definitions.

$lacadmin listener list

```
Listeners
Ident  Name          Type  Logging Level  Active  Code                                                  Description
-----  ------------  ----  -------------  ------  ----------------------------------------------------  -----------
2003   MQTTListener  MQTT  4              false   // Listener code goes here or check out
2004   MQTTResponse  MQTT  4              false   // Listener code goes here or check out

# listeners: 2
```

Export

$lacadmin listener export

```
[
 {
   "name": "MQTTResponse",
```

```
    "description": null,
    "is_active": true,
    "code": "// Listener code goes here or check out examples ( see top right dropdown menu ) \nlog.debug(\"Response from
published message \"+message);",
    "logging_level": 4,
    "provider_ident": 3,
    "ListenerParameters": [
     {
       "value": "lacresponse",
       "type_ident": 1,
       "description": null
     }
    ],
    "connection": {
     "@metadata": {
       "action": "LOOKUP",
       "key": "name"
     },
     "name": "MQTTConnection"
    }
  }
]
```


TeamSpace

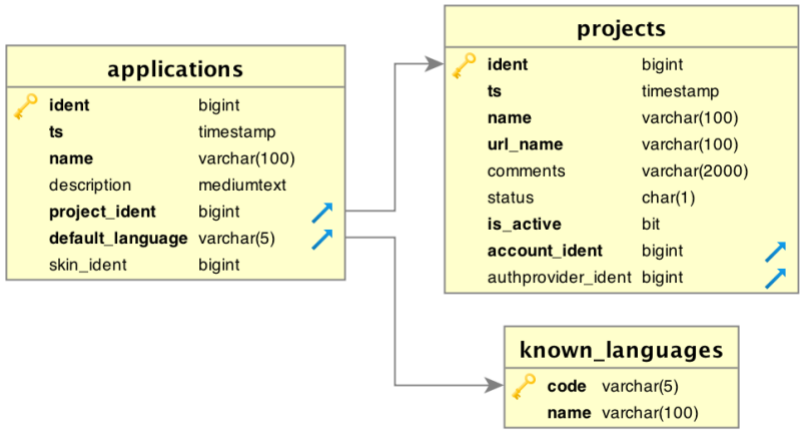
The new TeamSpace feature allows the 'sa' admin developer the ability to create new TeamSpaces (aka Accounts). The user interface will create the named TeamSpace, create a default user and password and optionally copy from an existing TeamSpace any custom authproviders and managed servers. The lacadmin teamspace command will let you list these teamspaces.

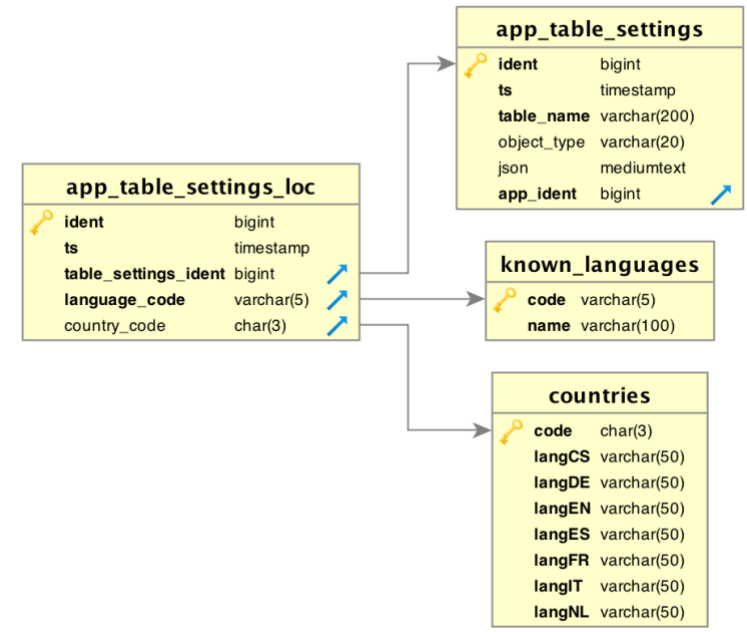
```
$lacadmin teampsace –help

Usage: teamspace [options] <list|export>

  Administer TeamSpace for current server.

  Options:

    -h, --help                output usage information
    --teamspace_name [name]   The name of the TeamSpace
```

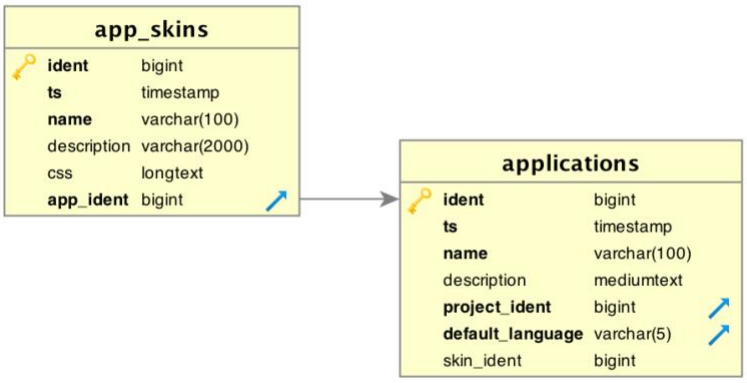
```
    --file [fileName]        optional: Name of file to import/export (if not
provided stdin/stdout used for export)
```

Applications

Applications refers to the Data Explorer module. By default, the system will generate dynamic views of tables, columns, relationship tabs. However, in author mode, the admin can change the order and visibility of columns, tabs, and formats for specific fields. The lacadmin command line tools for the Data Explorer meta data management will export the entire definition in a single JSON file.







Help

```
$lacadmin application --help
```

Usage: application [options] <list|delete|import|export>

Administer Data Explorer Applications.

Options:

```
  --ident [ident]           The ident of the specific project (see project list)
  --project_name [name]     The name of the project
  --url_name [name]         The name of the project
  --application_name [name] The name of the application
  --file [file]             optional: for import/export, the name of a file to read
  -h, --help                output usage information
```

List

```
$lacadmin application list
All Data Explorer Applications
Ident  Name        ProjectIdent  SkinIdent  Description
-----  ----------  ------------  ---------  -----------
8      Default app  2000          null
10     Default app  2013          null

# applications: 2
```

Export

```
$lacadmin application export –file APPLICATION_demo.json
```

Import

```
$lacadmin application import –file APPLICATION_demo.json
```

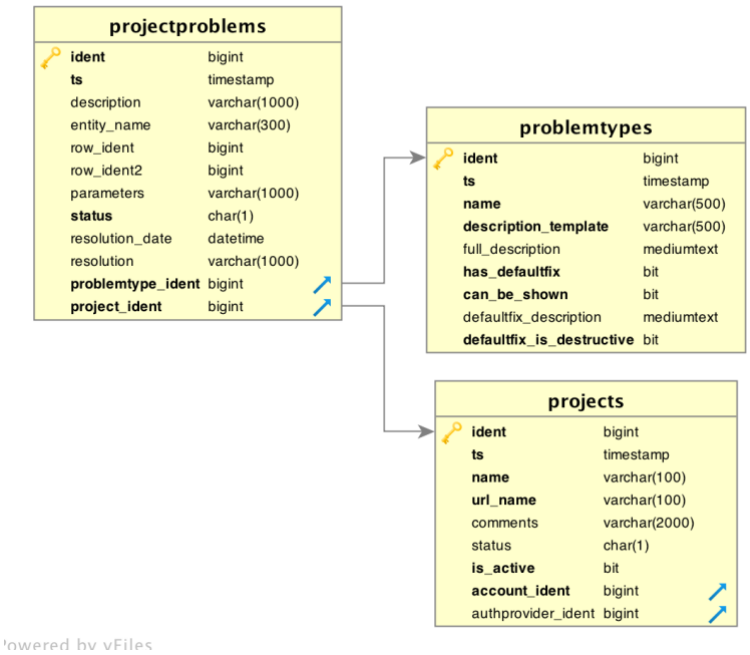
Other Tables

The Admin database also contains tables that may be viewed and provide important information. Using the NodeJS command line tool (`$npm install liveapicreator-cli –g`) and connecting using the admin url, username, and password you will be able to interact with these tables.

```
$lac login –u sa –p Password1 http://localhost:8080/rest/abl/admin/v2
```

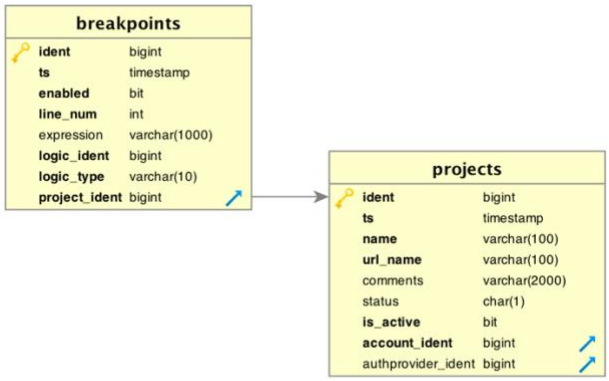
Project Problems (aka issues)

This is an internal table that the Live API Creator GUI uses to track issues when the verify button is executed.



Breakpoints

This is an internal table used by the debugger to track requests to put break points various objects (logic_type is rules, events, resources, functions, etc). Note; These settings are not persisted between server restart.




```
$lac get admin:breakpoints
```

Dbaseschema_metacache and dbaseschema_metacache_info

These are internal tables used by the server to cache SQL server meta information. When you reload your database schema, the metacache is reloaded. When you refresh from the browser, the cache is read instead of the SQL database.

Dbaseschema_role_strategies

Import Server License

The server license is required and can be imported from the command line:

```
$lacadmin license import –file file.json
```

To see your existing license, use:

```
$lacadmin license list
```

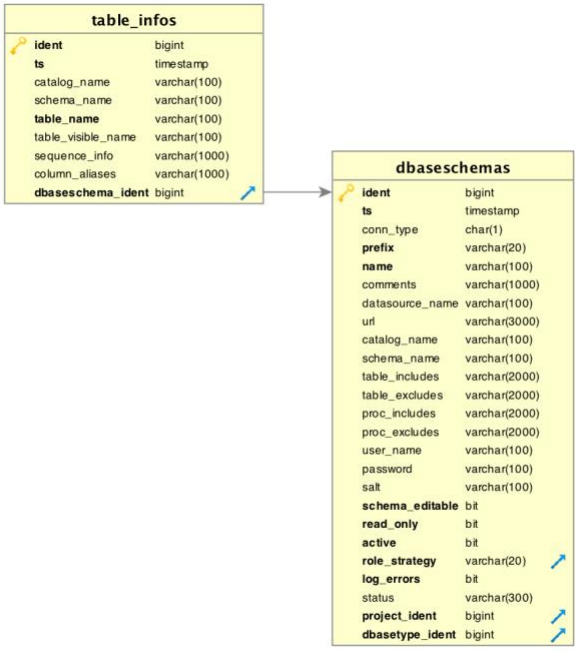
```
License
Company         Organization     location  license_type  Expiration
--------------  ---------------  --------  ------------  ------------------------
CA Technologies  Evaluation only  Anywhere  TRIAL         2017-02-18T15:59:59.000Z
```

Server Propertiesm

- prop_name prop_value
- admin schema version 20161121
- eula acceptance accepted ENU 2016-11-30T22:04:05Z

Table Info (Sequences)

This table is used to hold the sequence definition for each table/column – database specific. These can be set via the IDE.




```
$lac get admin:table_infos
```

Schema (Create)

If you have exported a schema or used the command line utility to generate a schema from Swagger, the command line can be used to create tables, columns, keys, and relationship on the current selected project API. Note: the prefix must be a managed server or the datasource is marked by isEditable flag as true. (In REST Lab, you can use /@schema/{prefix} to get your current schema definition for the datasource identified by the prefix name). See Manage Existing Schema documentation.

```
$lacadmin schema --help
```

Usage: schema [options] <create>

Create new table/columns/relationships using @schema format and managed data server datasource.

Options:

-h, --help output usage information
--project_ident [project_ident] The project ident that will be marked as used
--prefix [prefix] The datasource prefix used for export. Note for

```
                                                    import, the prefix must be marked as schema
                                                    isEditable
--ignoredbcolumntype [true|false]    (optional) The ignoredbcolumntype setting is used
                                                    when moving between database vendors
--ignoreprimarykeyname [true|false]  (optional) The ignoreprimarykeyname setting is used
                                                    when moving between database vendors
--ignoreconstraintname [true|false]  (optional) The ignoreconstraintname setting is used
                                                    when moving between database vendors
--skiprelationships [true|false]     (optional) If true, relationships will not be
                          created - default: false
--skiptablecreation [true|false]     (optional) If true, tables will not be created –
                          default: false
--file [fileName]                    [Optional] Name of file to import/export
```

Note: The flags may be necessary if you are trying to create a schema to a different database type. The schema contains column definition details that are database specific. If the flags are used, then the import creation will attempt to use approximate settings of generic_type (e.g. string, number, boolean, etc.) [as of v3.2 has new flags –skiptablecreation [true|false] and –skiprelationship [true|false] -these flags will allow import of data after the table creation and then only process relationships.)

```
$lacadmin schema create --prefix demo2 --file SCHEMA_myproject.json
```

End-user License Agreement (EULA)

This command returns true or false and indicates if the EULA has been accepted. If false – many of the lacadmin commands and IDE will not respond correctly until the EULA has been accepted.

```
$lacadmin eula accepted
true
```

Migrate (Repos)

The migrate utility will create a script which can be used to export repository libraries, auth providers, gateways, projects and their contents into specific files. This is only an admin script that can be modified for import into a target server. Note: the double dash directory flag is required and the directory must exist. (See GitHub EspressoLogicCafe/MigrationService).

```
$lacadmin migrate exportRepos --directory temp
```

```
lacadmin login -u admin -p <password> http://localhost:8080 -a migrate
lacadmin use migrate
lacadmin managedserver export --file 'temp/MANAGED_SERVERS.json'
lacadmin libraries export --ident 2001 --file 'temp/LIBRARY_TransformToWebHook.json'
lacadmin libraries export --ident 2000 --file 'temp/LIBRARY_B2BAuthProvider.json'
lacadmin authprovider export --ident 2000 --file 'temp/AUTHPROVIDER_AuthProviderFromDB.json'
lacadmin libraries export --ident 2002 --file 'temp/LIBRARY_AdminAuthProvider.json'
lacadmin project export --url_name b2bderbypavlov --file 'temp/PROJECT_b2bderbypavlov.json'
lacadmin project export --url_name test --file 'temp/PROJECT_test.json'
lacadmin project export --url_name xkdvx --file 'temp/PROJECT_xkdvx.json'
lacadmin project export --url_name demo --file 'temp/PROJECT_demo.json'
lacadmin project export --url_name sample --file 'temp/PROJECT_sample.json'
lacadmin project export --url_name b2bderbynw --file 'temp/PROJECT_b2bderbynw.json'
```

CA Live API Creator Command Line

The data itself can be viewed using the other command line utility **(`$npm install liveapicreator-cli -g`)**. This command allows the devops to connect any Live API Creator server as an end-user and issue commands to return project level data. You will need to full name of the server, port, and project url. (See EspressoLogicCafe/liveapicreator-devops for the compiled version).

Help

`$lac --help`

```
Usage: lac [options] [command]

Commands:

  login [options] <url>          Login to an API Server
  logout [options] [url]         Logout from the current server, or a specific server
  use <alias>                    Use the specified server by default
  status                         Show the current server, and any defined server aliases
  get [options] <resource>       Retrieve some data for the given resource/table/view
  post [options] <resource>      Insert some data
  put [options] <resource>       Update some data
  delete [options] <resource>    Delete some data
  describe [options] <resource>  Describe the specified resource, can be:
                                                    tables[/tablename], views[/viewname], resources, license,
                                                    serverinfo
  schema [options] <list>        Administer API project options for an account.

Options:

  -h, --help     output usage information

  -V, --version  output the version number
```

Login

The login command follows the same syntax as the admin command line but the server must point to a specific API project endpoint.

```
$ lac login -u demo -p Password1 http://localhost:8080/rest/default/demo/v1 -a demo
Logging in...
This server licensed to: CA Technologies license_type: TRIAL
License will expire: 2017-06-03T16:59:59.000Z
Login successful, this API key will expire on: 2016-12-30T02:37:22.858Z
$lac use demo
You are now using server http://localhost:8080/rest/default/demo/v1 as user demo
```

Status

```
$lac status
```

Defined aliases:

Alias	LAC Server	User
admin	http://localhost:8080/rest/abl/admin/v2	sa

You are currently logged in to CA Live API Creator server: http://localhost:8080/rest/abl/admin/v2 as user sa

GET -Help

The get command will retrieve the first N set of rows in a table compressed view. To see the raw JSON data, use the double-dash format (--format json) flag and the pagesize to get more data. Other switches can be used for filters, sorts, offset, nometa, and primary key. A Resource is the name of a table (e.g. main:Customer) or user resource (MyCustomers). Top level functions may also be called as resource endpoints.

```
$lac get --help
```

```
Usage: get [options] <resource>

Retrieve some data for the given resource/table/view
```

Options:

```
  -h, --help                    output usage information
  -k, --pk <pk>                 Primary key of the table or resource
  -f, --sysfilter <sysfilter>   Optional: sysfilter, e.g. "less(balance:1000)" or
                                equal(colname:value)
  -s, --sysorder <sysorder>     Optional: sysorder, e.g. "(balance,name desc)"
  -g, --userfilter <name>       Optional: userfilter, e.g. "userName(colname:value)
  -t, --userorder <name>        Optional: userorder, e.g. "myPaidOrders"
  -z, --pagesize <pagesize>     Optional: up to how many rows to return per level
  -o, --offset <offset>         Optional: offset for starting next batch
  -n, --nometa <true>           Optional: If true, no @metadata will be returned
  -m, --format <format>         Optional: format of output, either text (default), json
                                or compactjson
  --truncate <length>           Optional: truncate values at this many characters
                                (default 20)
  -j, --jsonfile <filename>     Name of file to write JSON output
  -a, --serverAlias <serverAlias>  Optional: alias of the server to use if other than the
                                current default server
```

GET Sample

If you do not specify the output – you will get a truncated view of some of the data and the first N rows. Use the –m (format) switch with the 'json' to get the output in full JSON format.

```
$lac get demo:customer
GET for demo:customer:

demo:customer/Alpha%20and%20Sons name:Alpha and Sons balance:4484 credit_limit:9000
demo:customer/Argonauts name:Argonauts balance:1858 credit_limit:2000
demo:customer/Baja%20Software%20Ltd name:Baja Software Ltd balance:635 credit_limit:785
demo:customer/Black%20Sheep%20Industries name:Black Sheep Indus... balance:76 credit_limit:10000
demo:customer/Bravo%20Hardware name:Bravo Hardware balance:2996 credit_limit:5000
demo:customer/Charlie's%20Construction name:Charlie's Constru... balance:1351 credit_limit:1500
demo:customer/Delta%20Engineering name:Delta Engineering balance:2745 credit_limit:15000
demo:customer/Echo%20Environmental%20Services name:Echo Environmenta... balance:2002 credit_limit:8000
demo:customer/Foxtrot%20Farm%20Supply name:Foxtrot Farm Supply balance:2957 credit_limit:3000
demo:customer/Golf%20Industries name:Golf Industries balance:3359 credit_limit:9000
```

demo:customer/Hotel%20Services name:Hotel Services balance:4481 credit_limit:5000

demo:customer/India%20Investigators name:India Investigators balance:5696 credit_limit:6000

demo:customer/Jack%20Trading%20Co. name:Jack Trading Co. balance:46 credit_limit:3000

demo:customer/Jill%20Exports%20Ltd. name:Jill Exports Ltd. balance:43 credit_limit:4500

demo:customer/Juliet%20Dating%20Inc. name:Juliet Dating Inc. balance:1297 credit_limit:1500

demo:customer/Kilo%20Combustibles name:Kilo Combustibles balance:6476 credit_limit:22000

demo:customer/La%20Jolla%20Ice%20Cream name:La Jolla Ice Cream balance:59 credit_limit:5000

demo:customer/Lima%20Citrus%20Supply name:Lima Citrus Supply balance:65 credit_limit:7500

demo:customer/Mike%20and%20Bob's%20Construction name:Mike and Bob's Co... balance:8409 credit_limit:14000

demo:customer/November%20Nuptials%20Wedding%20Co name:November Nuptials... balance:223 credit_limit:6800

more...

Request took: 36ms - # top-level objects: 20

$lac get demo:customer –m json –f DATA_Customer.json

POST

POST a json file for the given named resource.

$lac post --help

```
Usage: post [options] <resource>

Insert some data

Options:

  -h, --help                  output usage information
  -j, --json <json>           JSON for the data being inserted
  -f, --jsonfile <jsonfile>   Name of a file containing JSON to be inserted, or stdin
                              to read from stdin
  -m, --format <format>       Optional: format of output, either text (default), json
                              or compactjson
  -a, --serverAlias <serverAlias>  Optional: alias of the server to use if other than the
                              current default server
```

$lac post ‘demo:customer’ –j Customer.json

PUT

The put command requires a json file with the correct attributes for the given resource (e.g. table or normal resource). The PUT must have an @metadata tag with the 'checksum' attribute for each level.

```
$lac put --help
```

```
Usage: put [options] <resource>

Update some data

Options:

  -h, --help                  output usage information
  -k, --pk <pk>               Primary key of the table or resource
  -j, --json <json>           JSON string for the data being updated
  -f, --jsonfile <jsonfile>   Name of a file containing JSON to be updated, or stdin
                                                    to read from stdin
  -m, --format <format>       Optional: format of output, either text (default), json
                                                    or compactjson
  -a, --serverAlias <serverAlias>  Optional: alias of the server to use if other than the
                                current default server
```

DELETE

The delete syntax requires a primary key and checksum to identify a row and to ensure the data has not been modified from another source. The checksum is returned with every GET request using the format 'json'.

```
$lac delete -help
```

```
Usage: delete [options] <resource>

Delete some data

Options:

  -h, --help                  output usage information
```

```
-k, --pk <pk>                    Primary key of the object to delete
--checksum <checksum>            Optional: checksum for the object to delete, or
                                           "override".
-m, --format <format>            Optional: format of output, either text (default), json
                                           or compactjson
-a, --serverAlias <serverAlias>  Optional: alias of the server to use if other than the
                                 current default server
```

Describe

The describe feature is a short hand for the meta tags like @tables, @views, @resources, @fucntions.

```
$lac describe --help
```

```
Usage: describe [options] <resource>

Describe the specified resource, can be: tables[/tablename], views[/viewname], resources, functions, license, serverinfo

Options:

  -h, --help                     output usage information
  -a, --serverAlias <serverAlias>  Optional: alias of the server to use if other than the current default server
```

Describe Tables

```
$lac describe tables
```

All Tables

```
DB      Table
------- ------------------
demo    customer
demo    employee
demo    employee_picture
demo    LineItem
demo    product
demo    PurchaseOrder
demo    purchaseorder_audit
```

finance orders

finance promotions

Tables: 9

Export Schema & Swagger Doc

The schema command will list the tables for the selected project. The swagger command will export a Swagger 2.0 JSON file (@docs). The export option will output @schema/{prefix} to the console (or use the -–file switch to write the output to the file system.)

```
$lac schema --help
```

```
Usage: schema [options] <list|swagger|export>

Administer API project options for an account.

Options:

  -h, --help                    output usage information
  --prefix [name]               This is the datasource prefix for @schema
  --project_ident [project_ident]  The project ident that will be marked as used
  --file [fileName]             [Optional] Name of file to settings for import/export
```

List

```
$lac schema list
```

```
Tables
Prefix   Name                   Entity
-------  ---------------------  -------------------
demo     demo:LineItem          LineItem
demo     demo:PurchaseOrder     PurchaseOrder
demo     demo:customer          customer
demo     demo:employee          employee
demo     demo:employee_picture  employee_picture
demo     demo:product           product
```

demo demo:purchaseorder_audit purchaseorder_audit

finance finance:orders orders

finance finance:promotions promotions

Export Swagger @doc

```
$lac schema swagger --file swaggerDoc.json
```

Export @schema

```
$lac schema export --prefix demo --file schemaDemo.json
```

Live API Creator Node SDK

Another tool for the devops team is the NodeJS Live API Creator SDK ($npm install APICreatorSDK). This is used to write more complex scripts that require the use of promises or nested results.

Connect (username and password)

```
'use strict';
var apicreator = require('./APICreatorSDK');
var api = apicreator.connect('http://localhost:8080/APIServer/rest/default/demo/v1', 'demo', 'Password1');
```

Connect using APIKey

```
api = apicreator.connect('https://localhost:8080/APIServer/rest/default/demo/v1', 'readonly');
```

GET

```
api.endpoint('demo:customer').get().then(function (data) {
  console.log(data);
});
```

POST

```
var customers, newCustomer;
customers = api.endpoint('/customers');
var alphaCustomer = {
  name: "Alpha",
  credit_limit: "1234"
};

//POST
customers.post(alphaCustomer, params).then(function (txSummary) {
  console.log(txSummary); //an array of objects updated
});
```

Put/Delete

```
//GET first (need the checksum)
customers.get().then(function (data) {
  console.log(data); //an array which will now include customer: Alpha

  //objects returned include metadata specific to each record,
  //the most useful to us here is the endpoint href
  var alphaEndpoint = api.endpoint(data[0]['@metadata'].href);

  //PUT
        data[0].name = 'Alpha Updated';
        alphaEndpoint.put(data[0]).then(function(txSummary) {
          console.log(txSummary);
    });

  //DELETE
        alphaEndpoint.delete(data[0]).then(function(txSummary) {
          console.log(txSummary);
    });
});
```

Appendix – List API Project Contents

This is a sample MAC OSX Script, for Windows replace the # with REM and insert **`call in`** front of lacadmin (e.g. call lacadmin).

```
#! /bin/bash
# Generate the contents of an existing repository
SERVER=http://localhost:8080
PROJECT=demo
## Connect to a local server $SERVER and use API Project $PROEJCT
lacadmin logout -a local
lacadmin login -u admin -p Password1 $SERVER  -a $PROJECT
lacadmin use $PROJECT
lacadmin status
# Select a specific Project
lacadmin project use --url_name $PROJECT
lacadmin project list
lacadmin apioptions list
lacadmin datasource list
lacadmin libraries list
lacadmin authprovider list
lacadmin rule list
lacadmin apiversion list
lacadmin resource list
lacadmin relationship list
lacadmin token list
lacadmin role list
lacadmin user list
lacadmin namedsort list
lacadmin namedfilter list
lacadmin event list
lacadmin handler list
lacadmin topic list
lacadmin npa list
lacadmin gateway list
```

```
lacadmin snapshot list
lacadmin managedserver list
lacadmin function list
lacadmin license list
lacadmin eula accepted
lacadmin virtualkey list
lacadmin sequences list
#LOGOUT current connection
lacadmin logout -a $PROJECT
```

Appendix – Export

This is a sample MAC OSX Script, for Windows replace the # with REM and insert call in front of lacadmin (e.g. call lacadmin).

```
#! /bin/bash
# List and Export the contents of an existing repository
SERVER=http://localhost:8080
PROJECT=demo
mkdir demo
## Export from local server
lacadmin logout -a local
lacadmin login -u admin -p Password1 $SERVER -a $PROJECT
lacadmin use $PROJECT
lacadmin status
# Projects
lacadmin project list
lacadmin project use --url_name $PROJECT
lacadmin project export --url_name $PROJECT --file demo/demo.json
#API Settings
lacadmin apioptions list
lacadmin apioptions export --file demo/apioptions.json
# Data Sources
lacadmin datasource list
lacadmin datasource export --prefix demo --file demo/demo_ds.json
lacadmin datasource export --prefix finance --file demo/finance_ds.json
#Libraries - change the <ident> to the value from the list
lacadmin libraries list
lacadmin libraries export --ident 2001 --file demo/demo_libraries.json
#Auth Providers
lacadmin authprovider list
lacadmin authprovider export --ident 2001 --file demo/demo_authprovider.json
```

```
lacadmin authprovider export --name RESTAuthSecurityProviderCreateJS --file
demo/demo_RESTAuthSecurityProviderCreateJS.json

#Rules (--verbose will print out rules in lacadmin create format)

lacadmin rule list --verbose

lacadmin rule export --file demo/rules.json

#Resources

lacadmin resource list

lacadmin resource --ident 2001 export --file demo/resources.json

#Virtual Primary Keys

lacadmin virtualkey list

lacadmin virtualkey export --file demo/virtualkey.json

#Function

lacadmin function list

lacadmin application list

lacadmin listener list

lacadmin connection list

lacadmin timer list

lacadmin sequence list

lacadmin function export --file demo/function.json

lacadmin logout -a $PROJECT
```

Appendix 3 – Meta List

This is a sample MAC OSX Script, for Windows replace the # with REM and insert call in front of lacadmin or lac (e.g. call lacadmin, call lac).

```
#! /bin/bash
# Uses NodeJS and Live API Creator command line interface
# npm install liveapicreator-cli -g
# Live API Creator meta @ rest endpoints
## add  --format json for a full JSON response
SERVER=http://localhost:8080/rest/default/demo/v1
#echo 1
# Note that the URL contains the entire path to the project
lac login -u demo -p Password1 $SERVER -a demo
lac use demo
#Show the current license info (add --format json) for full EULA
lac get @license
#returns OK if server is up
lac get @heartbeat
# Show All Tables and columns for selected table
lac get @tables
#show details for a specific table
lac get @tables/nw:Customers
# Show All views and columns for selected view
lac get @views
lac get @views/nw:Current%20Product%20List
# Show All Resoures and attribute for selected resources (using ident)
lac get @resources
lac get @resources/2001
# Show All Store Proc and attribute for selected proc (using ident)
lac get @procedures
#lac get @procedures/somename
```

```
#Show the performance metrics for sql, rules, and admin SQL
#(add --format json) for detailed view
lac get @perf --format json
lac get @perf/sql?projectId=2047
lac get @perf/rules?projectId=2047
lac get @perf/adminSql?projectId=2047
#List of Rules
lac get @rules
#API Project settings
lac get @apioptions
#Information on the default auth provider
lac get @auth_provider_info/1000
# Swagger 2.0 doc format
lac get @docs
#Information from the Auth Provider
lac get @login_info
#Schema
lac get @schema/demo -m json
lac logout
```

Appendix 4 – Import

This is a sample MAC OSX Script, for Windows replace the # with REM and insert call in front of lacadmin (e.g. call lacadmin).

```
#! /bin/bash
# import Script for Northwind Jetty

## Logon to local  Jetty server (if WAR file use
##http://localhost:8080/APIServer)
lacadmin logout -a local
lacadmin login -u admin -p Password1 http://localhost:8080 -a local
lacadmin use local
lacadmin status
lacadmin project use --url_name nwindb2b
#Libraries - need to import these before import of JSON project
#We can create the Library and use an existing JS library.  To update / delete
and recreate.
lacadmin libraries create --name RESTAuthSecurityProviderCreateJS \
--short_name restauth --libtype javascript --ver 1.0 \
--file nw/RESTAuthSecurityProvider.js  \
--comments 'RESTAuthProvider js Demo '

lacadmin libraries import --linkProject  --file nw/auth_libraries.json
lacadmin libraries list

# Projects - this is the default NorthWind JSON project
#- if you import - load libraries first
#lacadmin project import --file nw/nwind.json
lacadmin project use --url_name nwindb2b
lacadmin project list

#API Settings [Optional]
lacadmin apioptions list
lacadmin apioptions import --file nw/apioptions.json

# Data Sources [optional] for other databases - set the password
lacadmin datasource list
#lacadmin datasource update --prefix nw --password password1 -- Jetty does not
use pw

#Auth Providers - lets create a new one and link it to the current project
lacadmin authprovider list
lacadmin authprovider create --name RESTAuthSecurityProviderCreateJS \
--createFunction RESTAuthSecurityProviderCreate \
--paramMap
logonApiKey=Lvnq9CYXN5oYoiToWGkN,loginBaseURL=http://localhost:8080/rest/defaul
t/nwind/v1/nw%3AEmployees,loginGroupURL=http://localhost:8080/rest/default/nwin
d/v1/nw%3ARegion \
--comments 'Uses NW Employees for REST Validation'
lacadmin authprovider linkProject --name RESTAuthSecurityProviderCreateJS
```

```
#Rules [optional] - this will export each rule in a single JSON file, but the -
#-verbose will output each rule for create
lacadmin rule list --verbose
#lacadmin rule import --file nw/rules.json

#Resources [optional] how to load a complete set of reseources
lacadmin resource list
#lacadmin resource import --file nw/resources.json

#close connections
lacadmin logout -a local
```

Appendix 5 – Schema and Data Export/Import

This is a sample script written in the APICreatorSDK that will generate a script to export @schema, data, projects, and then import the schema (tables first), data, then relationships.

```
var apicreator = require('./APICreatorSDK');
var urlquery = require('./urlutil');
//setup your server and source and target project information
var server = 'http://localhost:8080/APIServer';
var source_project = 'demo'; //Source Project url fragment
var target_project = 'test';// Target Project url fragment
var project_prefix = 'demo';//Source Project Datasource prefix
var target_prefix = "ldgiw"; //Managed Data Server Datasource Prefix on Target
Project

//connect to project and get list of all tables using prefix
var api = apicreator.connect(server + '/rest/default/' + source_project +
'/v1', 'demo', 'Password1');

console.log("#STEP 1 export schema");
console.log('lac login -u demo -p Password1 '+ server + '/rest/default/' +
source_project + '/v1');
console.log("lac schema export --prefix "+project_prefix +" --file
SCHEMA.JSON");

var tables = api.endpoint('@tables');
tables.get().then(function (data) {
      console.log("#STEP 2 Export Source Data as JSON Files");
      for(var idx = 0; idx < data.length; idx++) {
            var name =  data[idx].entity;
            var entity_prefix = data[idx].prefix;
            if(project_prefix === entity_prefix) {
                  console.log("lac get " + name + " -m json -z 999 -i 999999 -n
true -j DATA_"+name);
            }
      }
      console.log("lac logout");


      console.log("#STEP 3 Export source project ");
      console.log("lacadmin login -u admin -p Password1 "+server);
      console.log("lacadmin project use --url_name "+source_project);
```

```
      console.log("lacadmin project export --file
PROJECT_"+source_project+".json");
      console.log("lacadmin project use --url_name "+target_project);
      console.log("#if you are switching databases - include these flags in
Schema Create");
      console.log("# --ignoredbcolumntype true");
      console.log("# --ignoreprimarykeyname true");
      console.log("#STEP 4 PHASE 1 - SCHEMA CREATE TABLES ");
      console.log("#### OPTIONAL lacadmin project import --file
PROJECT_"+source_project+".JSON");
      console.log("#Create a managed datasource ans use this for your --prefix
below");
      console.log("lacadmin schema create --skipRelationships true --
skipTableCreation false --ignoredbcolumntype false --ignoreprimarykeyname false
--file SCHEMA.json --prefix "+target_prefix);
      //START IMPORT PROCESS HERE
      console.log('lac login -u demo -p Password '+ server + '/rest/default/' +
target_project + '/v1');

      console.log("#STEP 5 Import Data into Target Project");
      for(var idx = 0; idx < data.length; idx++) {
            var name =  data[idx].entity;
            var entity_prefix = data[idx].prefix;
            if(project_prefix === entity_prefix) {
                  console.log("lac post " + name + " -f DATA_" + name);
            }
      }
      console.log("#if you are switch databases - include these flags in Schema
Create");
      console.log("# --ignoreconstraintname true");
      console.log("#STEP 6 PHASE 2 - SCHEMA CREATE RELATIONSHIPS");
      console.log("lacadmin schema create --skipRelationships false --
skipTableCreation true --ignoreconstraintname false --file SCHEMA.json --prefix
"+target_prefix);
      console.log("lacadmin logout");
      console.log("lac logout");

})
```

OUTPUT

#STEP 1 export schema

lac login -u demo -p Password1 http://localhost:8080/APIServer/rest/default/demo/v1

```
lac schema export --prefix demo --file SCHEMA.JSON
#STEP 2 Export Source Data as JSON Files
lac get customer -m json -z 999 -i 999999 -n true -j DATA_customer
lac get employee -m json -z 999 -i 999999 -n true -j DATA_employee
lac get employee_picture -m json -z 999 -i 999999 -n true -j DATA_employee_picture
lac get LineItem -m json -z 999 -i 999999 -n true -j DATA_LineItem
lac get product -m json -z 999 -i 999999 -n true -j DATA_product
lac get PurchaseOrder -m json -z 999 -i 999999 -n true -j DATA_PurchaseOrder
lac get purchaseorder_audit -m json -z 999 -i 999999 -n true -j DATA_purchaseorder_audit
lac logout
#STEP 3 Export source project
lacadmin login -u admin -p Password1 http://localhost:8080/APIServer
lacadmin project use --url_name demo
lacadmin project export --file PROJECT_demo.json
lacadmin project use --url_name test
#if you are switching databases - include these flags in Schema Create
# --ignoredbcolumntype true
# --ignoreprimarykeyname true
#STEP 4 PHASE 1 - SCHEMA CREATE TABLES
#### OPTIONAL lacadmin project import --file PROJECT_demo.JSON
#Create a managed datasource ans use this for your --prefix below
lacadmin schema create --skipRelationships true --skipTableCreation false --ignoredbcolumntype false --
ignoreprimarykeyname false --file SCHEMA.json --prefix ldgiw
lac login -u demo -p Password http://localhost:8080/APIServer/rest/default/test/v1
#STEP 5 Import Data into Target Project
lac post customer -f DATA_customer
lac post employee -f DATA_employee
lac post employee_picture -f DATA_employee_picture
lac post LineItem -f DATA_LineItem
lac post product -f DATA_product
lac post PurchaseOrder -f DATA_PurchaseOrder
lac post purchaseorder_audit -f DATA_purchaseorder_audit
#if you are switch databases - include these flags in Schema Create
# --ignoreconstraintname true
#STEP 6 PHASE 2 - SCHEMA CREATE RELATIONSHIPS
lacadmin schema create --skipRelationships false --skipTableCreation true --ignoreconstraintname false --file SCHEMA.json --
prefix ldgiw
lacadmin logout
lac logout
```