

# Introduction to the Live API Creator Readiness Lab

## Goals

Practical exposure to

1. Extensibility – how do I extend rules, and utilize existing libraries
2. Debugging – how do I diagnose and correct bugs
3. Manageability – since there is no code gen, how do I address SCS (Source Code Control Systems) and DevOps deployment

## Scenario

Declarative Point and Click Services are helpful, but real-world projects require the services noted above.

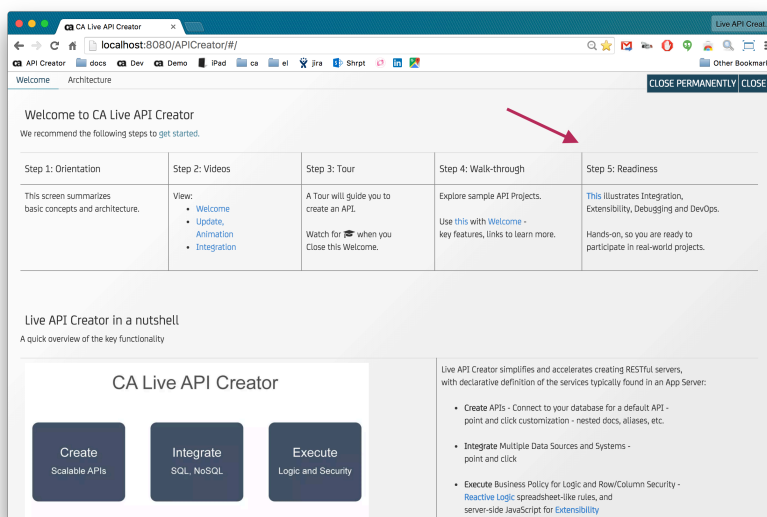
## Time

1 Hour

## Instructions:

**Before** taking this lab, complete the steps shown below on the Welcome page

- I. Get Oriented - Review Welcome Page
- II. Walk-throughs: Watch the Videos, and take the Welcome Walk-through
- III. Take the Tour
- IV. This exercise focuses on Step 5



## Lab 1: DevOps - Create the B2B API

---

<b>Goals</b>	Use script to create API from SCS artifacts
<b>Scenario</b>	The B2BScripts folder (containing this document) includes artifacts you might have put under Source Control System (SCS). Here, we deploy a running API from these SCS artifacts, and test it.
<b>Time</b>	5 minutes

---

### Instructions:

#### Install

Follow the [Install](#) instructions on the B2B document page.

#### Test

Follow the [Test](#) instructions on the B2B document page.

---

### Learnings

#### 1. [Scripted Deploy](#)

The script is a good example of one way to deploy SCS artifacts into production.

#### 2. [Northwind Schema: Readiness Data Model.png](#)

The B2BScripts folder contains a data model diagram you will find it helpful as you work through these exercises.

## Lab 2: Rules – no empty orders

---

<b>Goals</b>	Enter a rule
	Debug a rule
	Learn about Rule Execution Order
<b>Scenario</b>	A new User Story Behavior: no empty orders. That is, validate that <code>Orders.AmountTotal &gt; 0</code>
<b>Time</b>	20 minutes

---

### Instructions:

1. Create Rule(s) that implement the Behavior

2. Test pass / fail scenarios

Test with the Rest lab as you did above, for an Orders row *with* an Order Details item (exactly as done above), and an Orders Detail row with a quantity of 0 (just alter the json).

---

### Give it a try

An excellent first thought is to define a Validation rule on the Orders Table. As we'll see walking through the Learnings, the right solution is a Commit Validation Rule.

This is an intentionally tricky lab. Take a reasonable amount of time to try it. If you are successful, that's great. If not – not a problem... most folks find the attempt to be a good learning experience.

**But don't get frustrated!** After a reasonable amount of time, proceed to the Solution / Learnings, where we'll walk you through the solution, and why.

## Solution / Learnings

### 1. Rule Automatic Invocation

Rules often look very much like the business requirement, with a bit of syntactic sugar. For example, our validation on the Orders table:

```
return row.AmountTotal > 0
```

Note the introduction of the **row** variable. It's your [Object Model](#), created from the schema.

Note you must Activate the rule (e.g, Save and Activate).

OK, that's how it's *defined*, how is it *called*? In conventional systems, you have to not only create logic, but call it. [Reactive Logic](#) is **Declarative**: the system parses your logic for references, watches for changes to them, and reacts with proper rule execution (e.g., a derivation, a validation).

So.... there *is* no code to invoke the rule – invocation is automatic.

### 2. Log: shows all updated rows (and chaining)

A quick look at the available [Rule Types](#) might (and the problem statement), might suggest that a Validation rule is required. That's an excellent idea... but testing will reveal that it fails. Let's find out why.

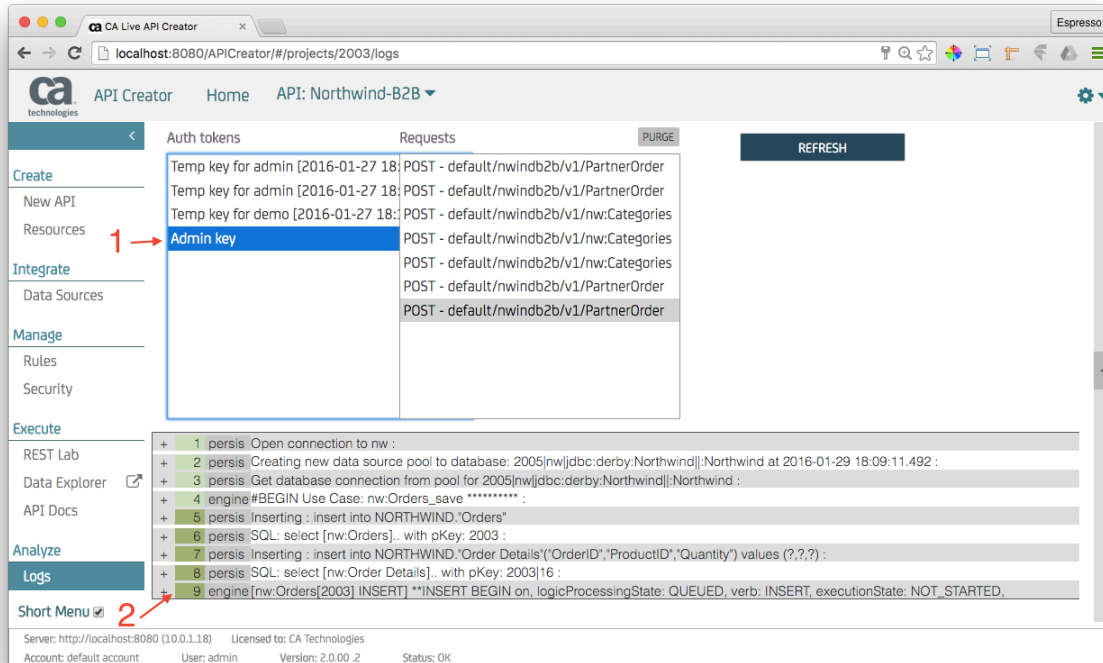
As above, we follow the [Test](#) instructions on the B2B document page. This reveals that:

- Orders of zero value are properly rejected. That's good.
- Orders *with* items (or Quantity 0) are also rejected. That's a bug.

For bugs, the **first thing to do** is check the [log](#) (you will need to select the Admin Auth Token – see the diagram below), which depicts

- Rule Execution - logs every rule that fires, including complete row state, with indenting to show multi-object chaining
  - Note: validation rules do not show in the log. Passed validations can be assumed, and failed validations result in exception and terminate the transaction
- SQL - use this to verify that SQL is expected

It's a bit subtle, but notice – we see the Orders row, but not the Details row. Hmm.



### 3. Rule Execution Order (commit after rule execution)

On reflection, this might make sense – when the Orders row is processed, the Details rows have not *yet* been processed, so our AmountTotal is 0. So, the validation fails.

What we need is a Validation that runs *after* the Order Details rows are processed, and the AmountTotal is adjusted. For that, we should use a Commit Validation.

### 4. Watch/React/Chain – multi-row chaining

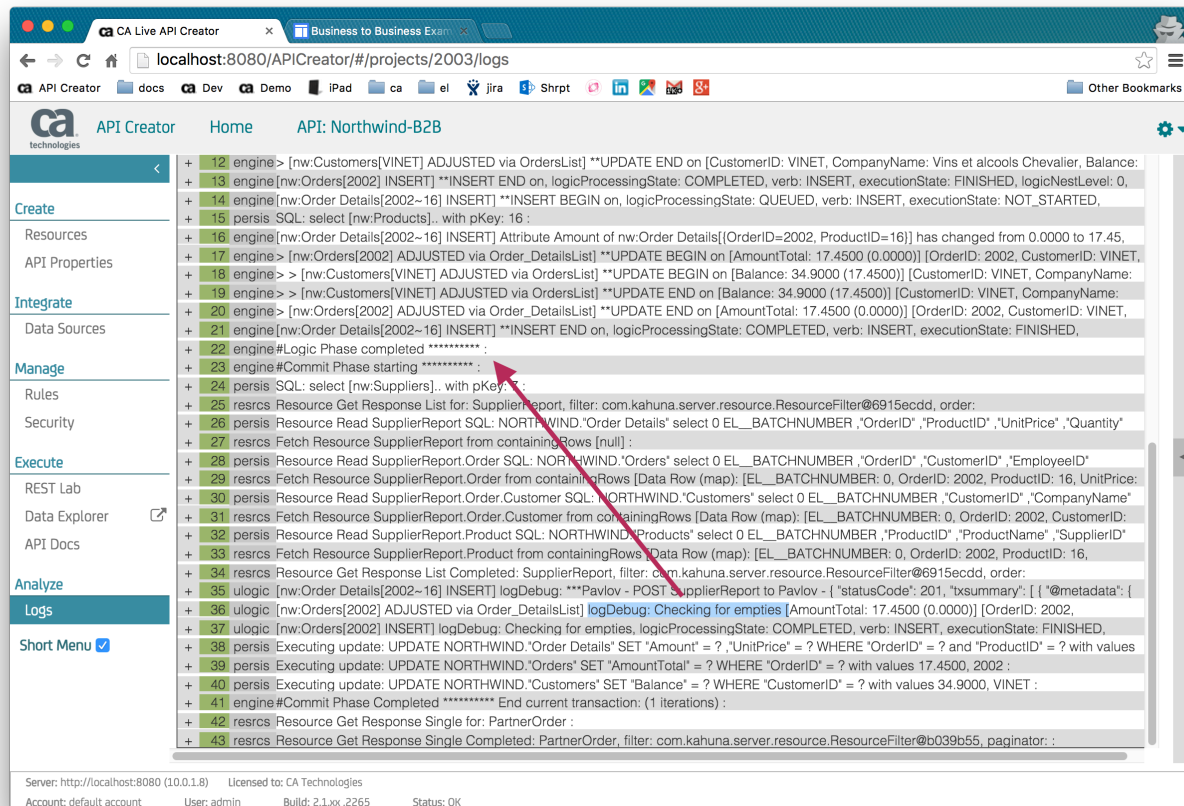
With this correction, it should work. Observe the log, and how the ident illustrates multi-row chaining, where each Order Detail adjusts the Orders AmountTotal. Then, after all the request rows are processed, our Commit Validation is executed, and works fine.

The log below illustrates operation. Note the critical difference between:

- Logic Phase processing – occurs as each row is processed from the request (the original 'plain' validation)
- Commit Phase processing – occurs *after* each row is processed from the request (our new *commit* time validation)

In this case, we have added a bit of debug code to the commit validation, since (to reduce log volume) passed validations don't appear in the log:

```
logicContext.logDebug('Checking for empties');
return row.AmountTotal > 0;
```



## 5. Final Note – what about Events

Many folks begin by using an Event rule. And, this can work (per the discussion above, a Commit Event). But it's not the best solution.

In most all cases, validations and derivations (formulas, sums, counts, parent replicates etc) are better solutions, since they automated your “watch” logic. That is, they only fire when their referenced data changes. This results in a meaningful reduction in code, and increase in quality.

## Lab 3: APIs for Mapping and Transformation Logic

---

<b>Goals</b>	Explore Custom Resources services for mapping and transformation
<b>Scenario</b>	There is a new agreement with Business Partners who place orders. They want the JSON to be <code>CustNbr</code> , instead of <code>CustomerNumber</code> .
<b>Time</b>	10 minutes

---

### Instructions:

#### 1. Revise JSON, Test the Fail scenario

Revise your test JSON to match the new API definition, and test as you did above. Note the failure.

#### 2. Change the API Definition for the revised agreement

Fix the API Definition (details noted in “Solution”, below).

#### 3. Test pass / fail scenarios

Re-test, should work with revised JSON.

---

### Give it a Try

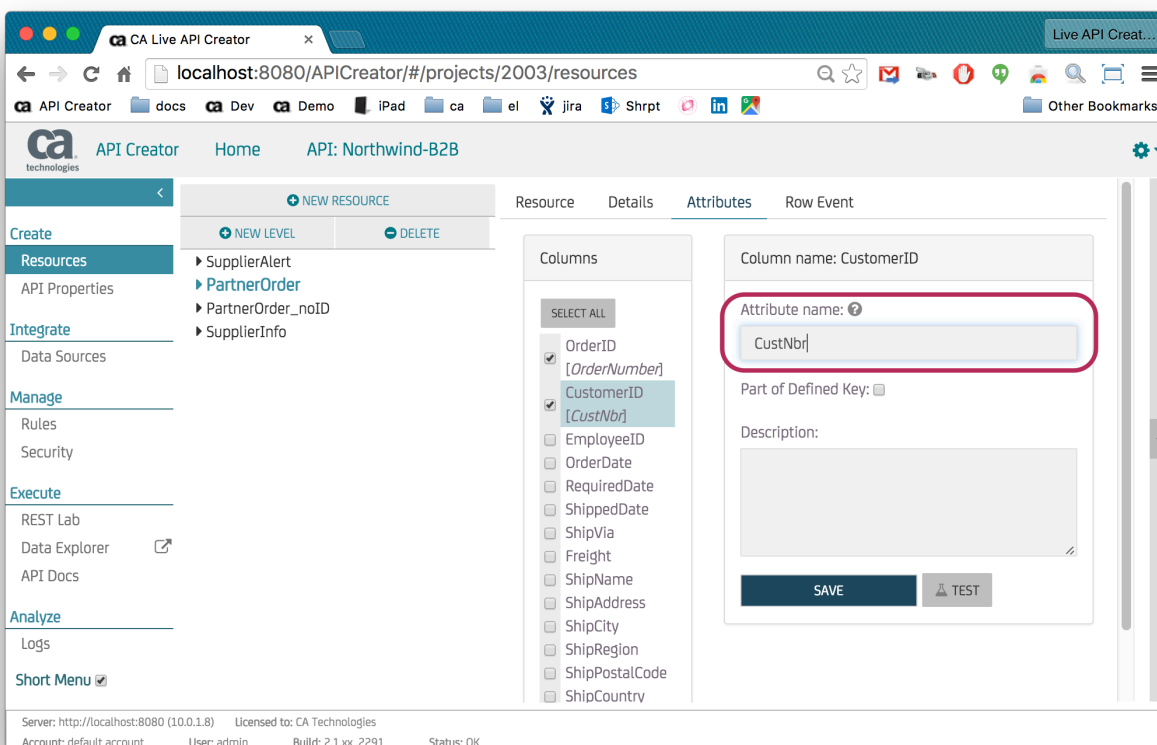
Just alter the Attribute Alias for the Resource `PartnerOrder`. The alias is available under the Attributes tab: Attribute Name.

---

### Solution / Learnings

#### 1. Resource Attribute Alias for Mapping / Transformation Logic

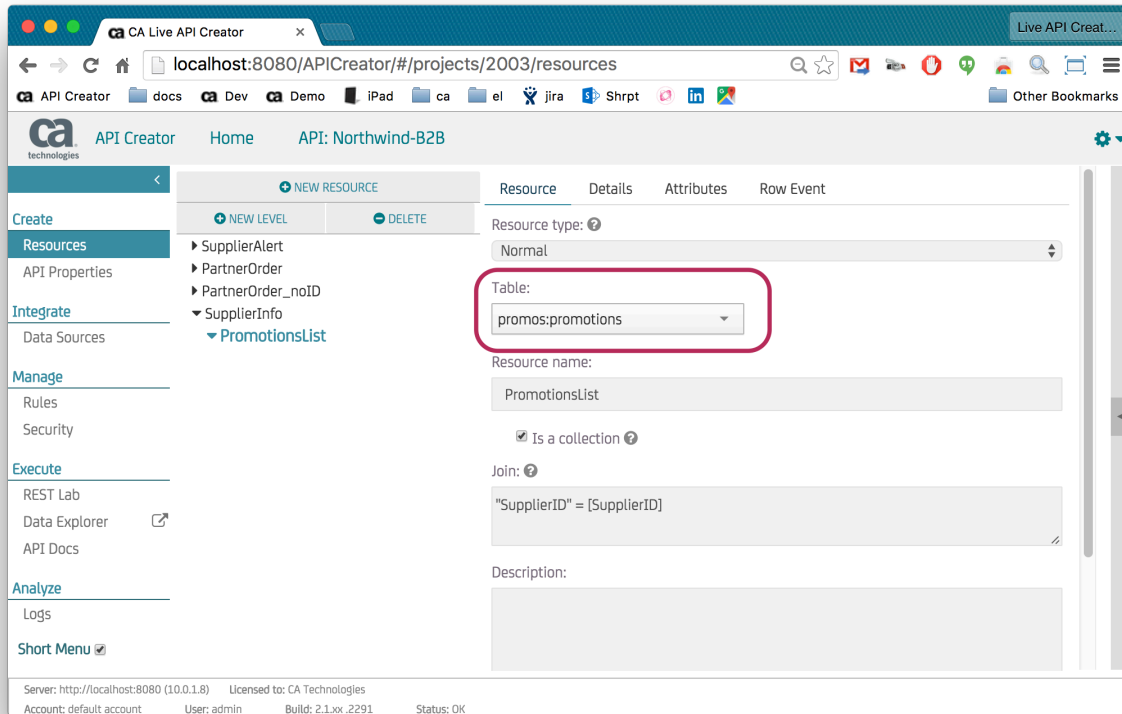
Public API definitions should not be constrained by your schema. Custom Resource definition supports the selection / renaming of Table and Attributes. This is often referred to as Mapping and Transformation logic.





## 2. Multiple Databases

APIs are often used to integrate multiple databases. Your API can connect to multiple databases, define relationships between tables across databases, and build Resources across databases. This resource illustrates providing information to Suppliers on Promotions, drawn from 2 different databases (check out the Data Sources).



## Lab 4: Extensibility

---

Goals	Explore JavaScript Extensibility
Scenario	Our team has introduced a re-usable pattern for creating resource instances to Post to partners. Make the current solution re-usable.
Time	15 minutes

---

### Instructions:

#### 1. Locate the Logic for Supplier Alert

This is the logic that notices orders for Products supplied by Pavlov, and Posts a message to alert Pavlov.

This is already implemented – under Rules, click the Topic Tag **Supplier Alert** (it's red).

#### 2. Observe the Event reference to the re-usable function

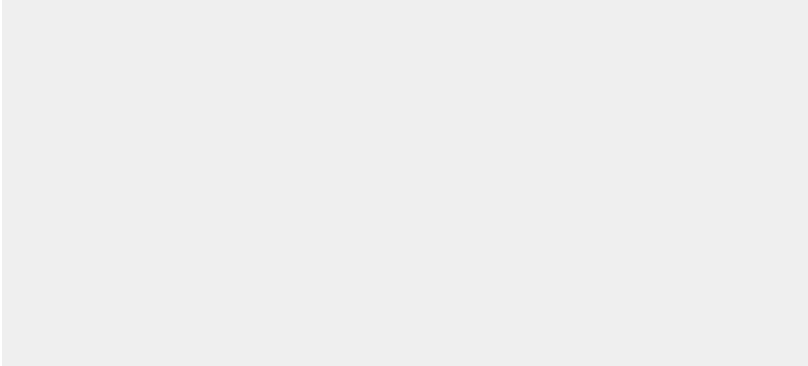
Replace the existing code with the following:

```
if (row.FK_Order_Details_Products.FK_Products_Suppliers.CompanyName == "Pavlova, Ltd." ) {  
    var response = postResourceTo("SupplierAlert",  
        { sysfilter: "equal(OrderID:'' + row.OrderID + '')" },           // filter  
        req.baseUrl.replace("nwindb2b","pavlov") + "v1/SalesReports",    // supplier URL  
        { headers: { Authorization: "CALiveAPICreator supplier:1" }});    // supplier auth  
    logicContext.logDebug('***Pavlov - POST SupplierReport to Pavlov - ' + response );  
}
```

#### 3. Change the Re-usable Solution

Alter **postResourceTo.js** to add a debug message, like this:

```
log.debug("ok, using re-usable solution");
```



4. Deploy
5. Test
6. Fix your error

---

## Give it a try

Your B2BScripts folder contains the source code: `postResourceTo.js`.

---

## Solution / Learnings

Let's walk through the solution, with an intentional error to explore the debug cycle.

### 1. Topics

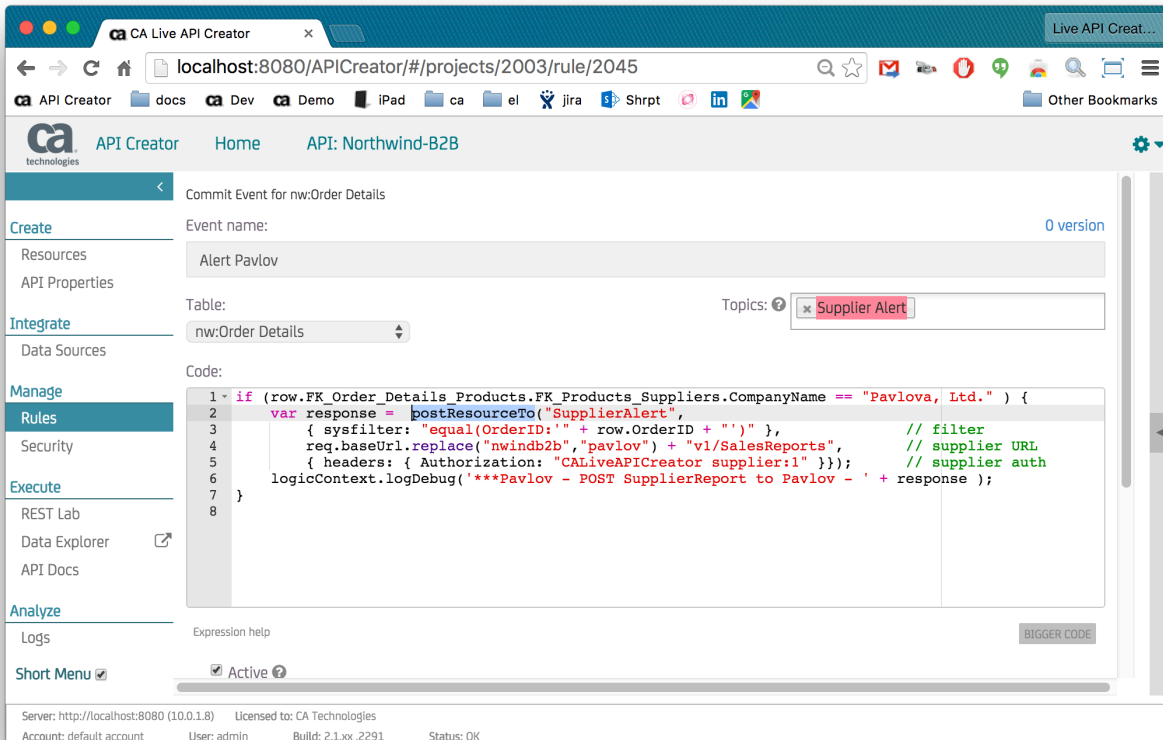
Topics enable you to capture *why* your logic is defined. They can be used to express (and link to) the Behaviors (Acceptance Criteria) for a User Story, here, Supplier Alert.

### 2. Events

Click the link to see the Event Rule.

Events enable you to add JavaScript, which operates in concert with declarative, spreadsheet-like logic.

Alter the event using the code above. It will look like this:



### 3. Loadable Libraries

Before you changed it, the logic was ‘in line’. Your change activated the use of an Loadable Library, provided so you can provide re-usable solutions to generic patterns.

Change the code to add the log.debug code shown above:

```
log.debug("ok, using re-usable solution");
```

But this only changes it on disk. You need to update the API Server. To do this, [reload your library](#) (you will need to disable Short Menus).

### 4. Test

Test as before using the Rest Lab - follow the [Test](#) instructions on the B2B document page. Note this logic is an Event, so only applies to update (not get) requests.

This will fail, since the `log` object is not defined for the Logic Library. We need to pass it from the event (it's part of the [JavaScript Context](#) passed to your event).

## 5. Fix, and re-load library

Add the argument to the event, so it looks like this.

```
if (row.FK_Order_Details_Products.FK_Products_Suppliers.CompanyName == "Pavlova, Ltd." ) {
    var response = postResourceTo("SupplierAlert",
        { sysfilter: "equal(OrderID:'" + row.OrderID + "')" },           // filter
        req.baseUrl.replace("nwindb2b","pavlov") + "v1/SalesReports", // supplier URL
        { headers: { Authorization: "CALiveAPICreator supplier:1" }}),
    log); // supplier auth
    logicContext.logDebug('***Pavlov - POST SupplierReport to Pavlov - ' + response );
}
```

Also fix the loadable library.

```
// this sample illustrates that you can Load Java and JavaScript Libraries, and call them from your
rules
// for instructions, please see: http://ca-doc.espressologic.com/docs/logic-designer/create/api-
properties/logic-libraries

function postResourceTo ( aResourceName, anOptions, aTargetUrl, aSettings, log) {
    var resourceResponse = SysUtility.getResource(aResourceName, anOptions); // resource
    provides name mapping
    var response = SysUtility.restPost(aTargetUrl, null, aSettings, resourceResponse[0]);
    log.debug("hello");
    return response;
}
```

Don't forget to re-load the library. (For in-line logic, there is no re-deploy required).

The test should succeed now.

## Lab 5: Data Explorer

---

<b>Goals</b>	Data Explorer – Author Mode
<b>Scenario</b>	For testing and back-office data maintenance, we want to customize the default User Interface provided by Data Explorer.
<b>Time</b>	10 minutes

---

### Instructions:

#### 1. Make Suppliers Region Blue, not Red

Start the Data Explorer, and Select the Supplier table. Click the **second Suppliers** row, and note the red Region in the upper right. (The first row has no Region, due to a “hide” expression you can explore below).

Make this Region field blue, not red.

#### 2. Change Customer Group Name Contact Info → Contact Information

Abbreviations are not so helpful to new folks.

---

### Give it a try

Use Data Explorer Author Mode

---

### Solution / Learnings

Let’s walk through the solution

#### 1. Start Data Explorer

Note you can start the Data Explorer *in*, or *outside*, the current window.

**Tip:** experienced users often utilize multiple Browser tabs, e.g., for logic, log, and Data Explorer to enter test data.

## 2. Select the Suppliers Table, select a Supplier row

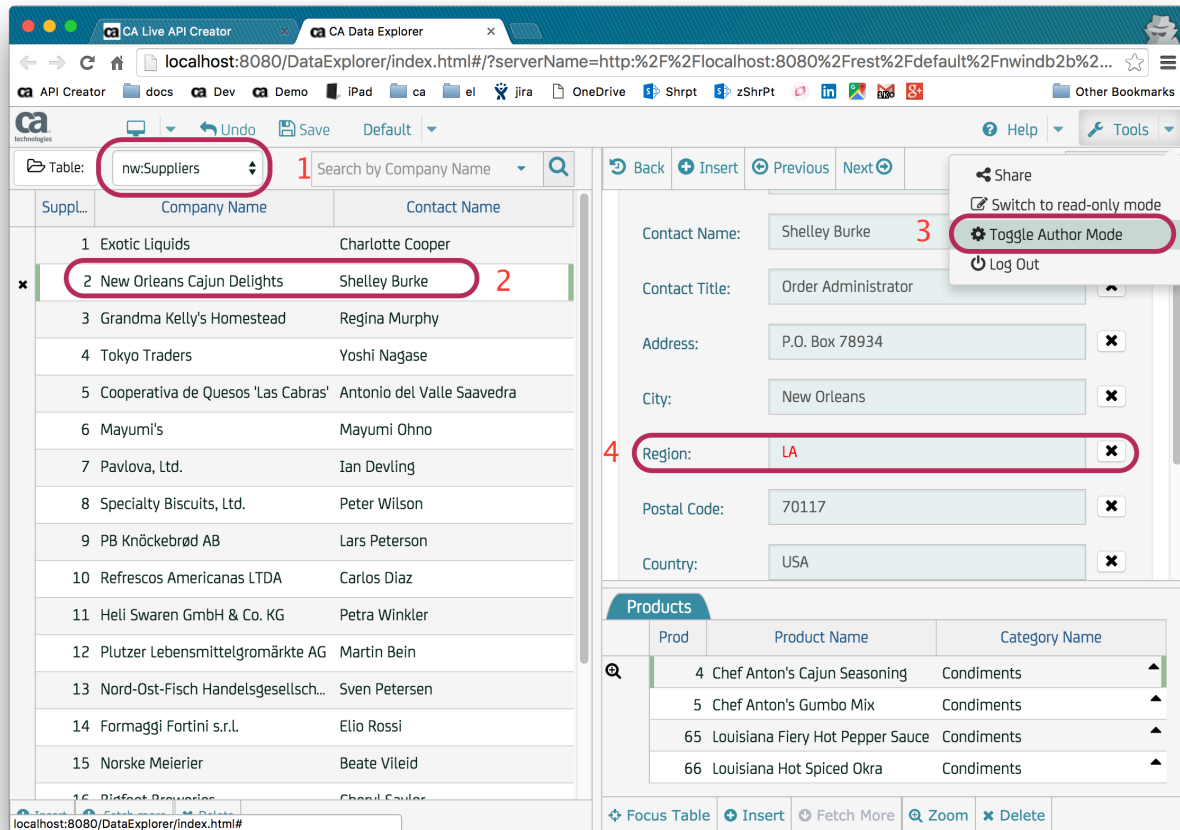
Click the second Suppliers row (step 2 below). (The first row has no Region, due to a “hide” expression you can explore below).

Note the upper right area has a Region field (see 4, below).

## 3. Enter Author Mode

Author Mode enables you to customize the look and feel. Use the Tools Menu, step 3, below.

You will need to enter a User Name – use admin.



#### 4. Click the Gear for the Region field

This appears after you've entered Author Mode.

#### 5. Alter the Expression Definition for blue

Observe you can define multiple expressions, e.g., one for color, one for hiding as illustrated in this example (and the doc).

#### 6. Select the Customers Table

And click one of the customers.



## 7. Click the Book Icon

This is the book ***on the right*** (not on the grid).

## 8. Click the Column Order Tab

You can use this screen to define Groups, their names, and what attributes are shown in each group. The documentation link above illustrates how to do this.

## 9. Change the text

Change “Contact Info” to “Contact Information”.

## Lab 6: Auth Provider

---

<b>Goals</b>	Understand how to build and deploy an Authentication Provider
<b>Scenario</b>	We want to alter the Auth Provider so that demo and admin do not require a password.
<b>Time</b>	10 minutes

---

### Instructions:

#### 1. Locate the Auth Provider

It is in your B2BScripts folder: `RESTAuthSecurityProvider.js`

This Auth Provider implements 2 ways to authenticate a user:

- Row exists in the `Employees` table
- Demo and admin are authorized by default (so Developers won't have to remember user names)

#### 2. Alter the Auth Provider

Be sure to add a debug statement so you can verify you are executing your altered code – look for “Lab test OK”, and remove the comment so the code is enabled.

#### 3. Deploy the altered Auth Provider

Use the same procedure as above.

#### 4. Test

Test your changes using the Data Explorer, and check the Server Console for your debug message. (The Server Console is the Command/Terminal window used to start the Jetty-based API Server).

## Solution

Follow the steps below.

## Learnings

Let's walk through the solution, with an intentional error to explore the debug cycle.

### 1. Auth Providers

Auth Providers are implemented as Loadable Libraries. Once you load and designate them, the system will invoke them when clients Create an Auth Token. This is what the Data Explorer does to begin a session.

In a production system, your Auth Provider might interface to Oauth, LDAP, AD, database tables, etc.

### 2. Alter the Auth Provider

As described above, it should look something like this:

```
result.authenticate = function authenticate(payload) {

  out.println("Authentication called...");
  var roles = [];
  var errorMsg = null;
  var resetPasswordURL = null;
  var forgotPasswordURL = null;
  var customDataHREF = [];
  var params = null;
  var settings = {
    headers : {
      'Authorization' : 'CALiveAPICreator ' + configSetup.logonApiKey + ':1'
    }
  };

  try {
    if (payload.username == 'admin' || payload.username == 'demo') {
      out.println("Authentication - default admin/demo user - good to go..");
      out.println("Lab test OK...");
      roles = ['Full access']; // || HARD CODED FOR DEMO (we even ignore the pwd)
      errorMsg = null; // if one role is found then we are good to return
    }
  }
}
```

### 3. Deploy it – upload the altered library

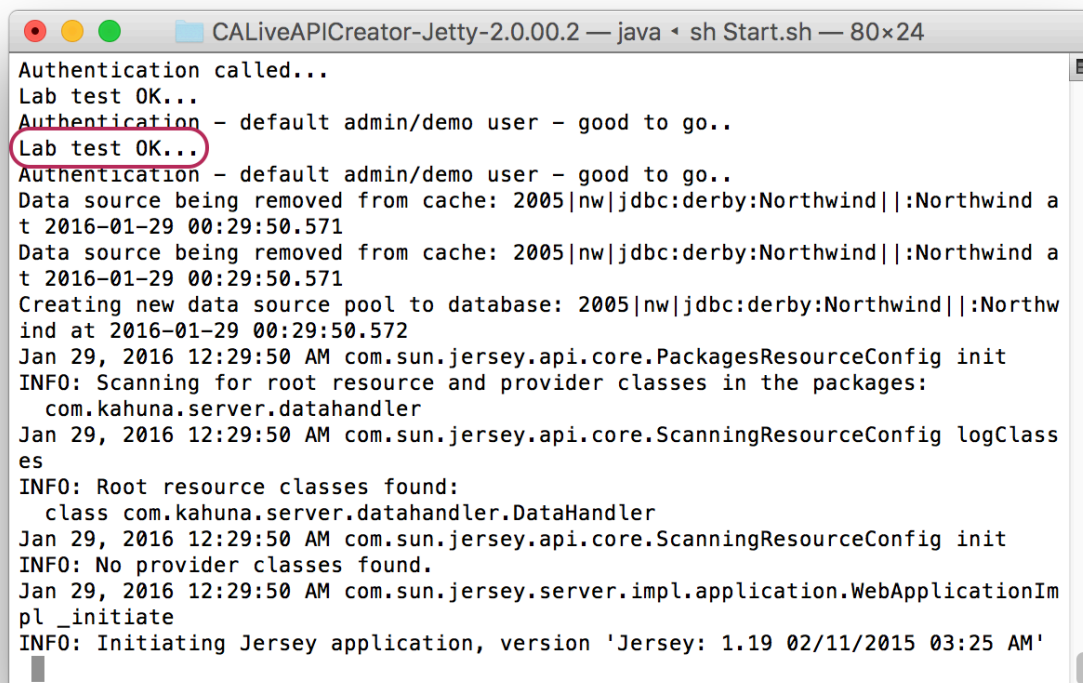
You can re-deploy the altered Auth Provider using the [API Creator](#).

### 4. Test with Data Explorer

You will often test with the Rest Lab, using a **pre-created** Auth Token. For example, Admin key is pre-defined for Northwind-B2B. Such [manually created Auth IDs](#) are typically used for development.

This by-passes the Auth Provider protocol, so you can't test your Auth Provider using the Rest Lab. Use the Data Explorer:

1. Logout (gear at upper right)
2. Login
3. Verify it works if you supply a silly password.
4. See your debug output, using the Terminal window for the API Server:



```

CALiveAPICreator-Jetty-2.0.00.2 — java ◀ sh Start.sh — 80x24
Authentication called...
Lab test OK...
Authentication - default admin/demo user - good to go..
Lab test OK...
Authentication - default admin/demo user - good to go..
Data source being removed from cache: 2005|nw|jdbc:derby:Northwind||:Northwind a
t 2016-01-29 00:29:50.571
Data source being removed from cache: 2005|nw|jdbc:derby:Northwind||:Northwind a
t 2016-01-29 00:29:50.571
Creating new data source pool to database: 2005|nw|jdbc:derby:Northwind||:Northw
ind at 2016-01-29 00:29:50.572
Jan 29, 2016 12:29:50 AM com.sun.jersey.api.core.PackagesResourceConfig init
INFO: Scanning for root resource and provider classes in the packages:
    com.kahuna.server.datahandler
Jan 29, 2016 12:29:50 AM com.sun.jersey.api.core.ScanningResourceConfig logClass
es
INFO: Root resource classes found:
    class com.kahuna.server.datahandler.DataHandler
Jan 29, 2016 12:29:50 AM com.sun.jersey.api.core.ScanningResourceConfig init
INFO: No provider classes found.
Jan 29, 2016 12:29:50 AM com.sun.jersey.server.impl.application.WebApplicationIm
pl_initiate
INFO: Initiating Jersey application, version 'Jersey: 1.19 02/11/2015 03:25 AM'
```

## Lab 7 – Advanced Feature survey

---

Review the [Advanced Topics](#). This illustrates additional extensibility options for reformatting JSON, and using metadata tags to trigger advanced processing.