Security Controls and Matillion: Enable User Auditing (Part 1 of 3)

In this three-part technical article series, we will demonstrate how common data-warehouse design patterns can be leveraged, along with a few features in Matillion ETL, to enable a critical Security Detective Control: user auditing.

We will take a detailed look at using Matillion ETL to implement Dimensional Modeling design patterns for advanced analytics. Matillion v1 API endpoints are used as the primary source of data and also help to demonstrate how to implement common API integration requirements in Matillion ETL.

*All **included Jobs** and Matillion ETL screenshots were built using **Matillion ETL for Snowflake** (AWS), version 1.39.10.*

Enable User Auditing, Part 1: Using the Matillion v1 API and API Integration with Matillion ETL

- Using Matillion API v1
- Paging logic in an API Profile
- Passing values to an API Query at runtime

In the referenced **Matillion ETL jobs**, we use **Matillion API v1** endpoints as the source of data to load into our cloud data warehouse – specifically, the **userconfig** and **audit** endpoints. Note that **Audit Log** information is a Matillion ETL **Enterprise Mode** feature and for that reason, this API endpoint will only be available when running Matillion ETL on a Large or X-Large instance type.

Matillion API Setup

User Setup

In order to use the Matillion API endpoints mentioned above, you must setup a **Matillion User** that has "API" access. This Matillion User must also have "Admin" access in order to access Audit Log information.

API Profile Setup

The article on **Connecting to Any REST API** walks through the steps of integrating with an API endpoint as a source of data through Matillion ETL. The specific **API Profiles** used for this exercise are:

Userconfig API

This API endpoint returns all internal users that exist in Matillion ETL, along with True/False boolean indicators of "Admin" or "API" access. **Get the Userconfig API Profile RSD**

Audit API

This API endpoint allows for the download of Audit Log information in JSON format. Notice that this API also pages its results based on "offset" and "limit" query parameters.

```
<rsb:script xmlns:rsb="http://www.rssbus.com/ns/rsbscript/2" xmlns:xs="http://www.w3.org/2001/XMLS
            Column Definitions to specify column behavior and use XPaths to extract column values from J
 >  <rsb:info title="audit" desc="Generated schema file." xmlns:other="http://www.rssbus.com/ns/rsbscrip
   You can modify the name, type, and column size here.  The name must match the values in the rsb:scrip
                          >    <attr name="commandType" xs:type="string"  readonly="false"
                other:xPath="commandType"  />    <attr name="id"          xs:type="integer" readonly="fal
```

```
other:xPath="id" /> <attr name="specifier" xs:type="string" readonly="false"
other:xPath="specifier" /> <attr name="timestamp" xs:type="long" readonly="fals
other:xPath="timestamp" /> <attr name="user" xs:type="string" readonly="fals
other:xPath="user" /> <input name="result_offset" xs:type="string" default="100" /> <input n
desc="A system column used for paging. Do not change." /> </rsb:info> <!-- Column XPaths are relative
that splits the JSON into rows. --> <rsb:set attr="RepeatElement" value="/list" /> <!-- The GET metho
SELECT. Within the script block, you can see the URI modified to append a query string parameter. The re
are pushed to the schema's output. See SELECT Execution for more information. --
> <rsb:script method="GET"> <rsb:set attr="EnablePaging" value="TRUE"
/> <rsb:check attr="Rows@Next"> <rsb:set item="userns" attr="offset" value="[_input.Ro
add([_input.result_offset]) ]" /> <rsb:set attr="uri"
value="http://127.0.0.1:8080/rest/v1/audit/export?limit=[_input.result_offset]&amp;offset=[usern
/> <rsb:else> <rsb:set attr="uri" value="http://127.0.0.1:8080/rest/v1/audit/export?limit=[_inp
/>" <rsb:set item="userns" attr="offset" value="0"
/> </rsb:else> </rsb:check> <rsb:call op="jsonproviderGet"> <rsb:set attr="R
value="[userns.offset]" /> <rsb:push/> </rsb:call> </rsb:script> </rsb:script>
```

Get the Audit API Profile RSD

API Paging

The Audit API Profile includes paging logic that differs slightly from the paging logic described in our article on **Paging with the API Query Component.** In the referenced article, the URI for the next page of results is included in the first page's response. Getting information from response data of the current page is a common API paging design pattern. The paging logic in the Audit API is another common API paging design pattern, where an incremented value is passed as a query parameter in the URI. In this instance, we are passing an "offset" and "limit". The "limit" represents the number of results to return in a page and the "offset" represents at which result to start for the requested page of results.

Here, we are using an Input attribute, "result_offset", to define the number of rows to return per page, giving it a default value of 100 within the API Profile definition. Because this is an Input attribute, we can also change this value at runtime. See our article on **Using Parameters with API Profiles** for more details on this topic.

```
<input name="result_offset" xs:type="string" default="100" />
```

For the first page of results, we simply return X number of rows of data, where X is defined by the aforementioned result_offset input attribute.

```
                                    <rsb:set attr="uri"
    value="http://127.0.0.1:8080/rest/v1/audit/export?limit=[_input.result_off
```

After defining the URI for the first page of results, we initialize a user namespace attribute, named "offset", with a value of 0.

```
                        <rsb:set item="userns" attr="offset" value="0" />
```

The "offset" user namespace attribute is used to dynamically set the "offset" query parameter in the URI for subsequent pages of results. We do this by performing some math on the "result_offset" input attribute value to the current value of the "offset" user namespace attribute for every iteration of a page.

```
<rsb:set item="userns" attr="offset" value="[_input.Rows@Next | add([_input.res
                                                    />
```

Similar to our article on **Paging with the API Query Component**, the "Rows@Next" input attribute is used in support of assigning the current value of the "offset" user namespace attribute.

```
                        <rsb:set attr="Rows@Next" value="[userns.offset]" />
```

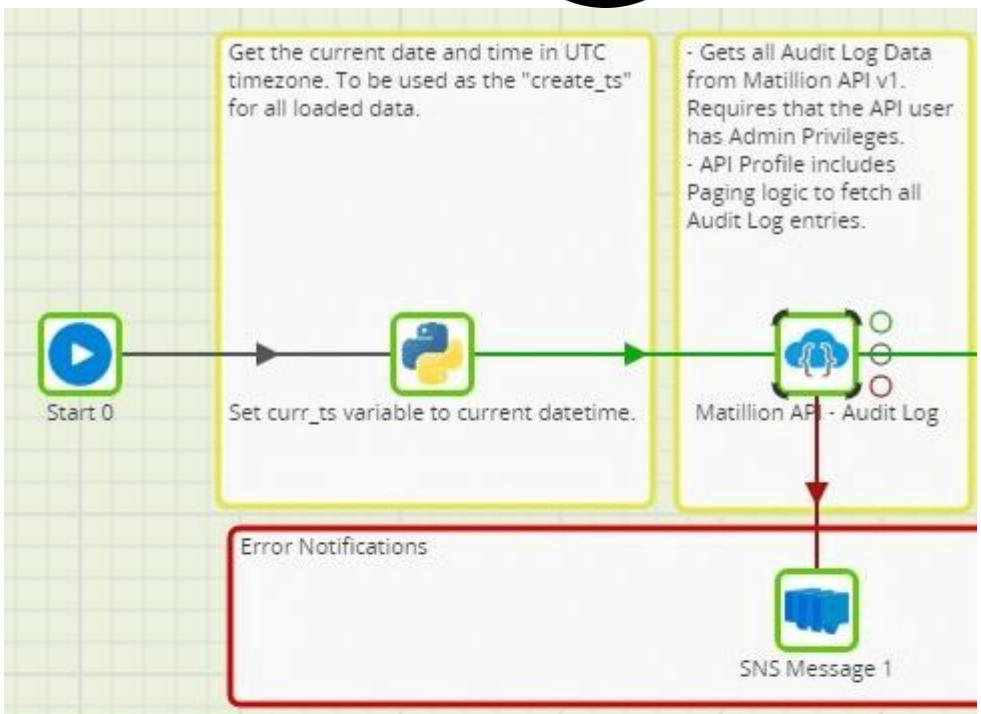
The "offset" user namespace attribute is then referenced in the URI for subsequent pages of results.

```
                                                <rsb:set attr="uri"
value="http://127.0.0.1:8080/rest/v1/audit/export?limit=[_input.result_offset]&
                                                    />
```

***Note**: The URI must be fully URL encoded, thus the ampersand (&) that is a delimiter between query parameters is represented as:* **&**

API Query Setup

Once the API Profiles have been setup to use the Matillion API v1 endpoints as a source, they can be used in an Orchestration Job by configuring the **API Query** component to use the appropriate API Profile as a source.



Properties | Sample | SQL | Export | Help

API Query — OK

Name	Value		Status
Name	Matillion API - Audit Log	...	OK
Basic/Advanced Mode	Basic	...	OK
Profile	Arawan-MatillionAPI	...	OK
Connection Options	AuthScheme, Basic, User , api-...	...	OK
Data Source	audit	...	OK
Data Selection	commandType, id, specifier, ti...	...	OK
Data Source Filter	result_offset, is, Equal to, 100	...	OK
Combine Filters	And	...	OK
Limit		...	OK
Primary Keys		...	OK
Warehouse	[Environment Default]	...	OK
Database	[Environment Default]	...	OK
Schema	[Environment Default]	...	OK
Target Table	stg_mtln_api_audit	...	OK
Staging	Snowflake Managed	...	OK
Load Options	On, Off, On, On	...	OK

Authentication

Assuming you have configured your Matillion ETL instance to use **Internal user authentication,** when setting up the API Query component to use a Matillion API as a source of data, you must pass the related authentication information as Connection Option parameters. The following are the Connection Option parameters that must be defined:

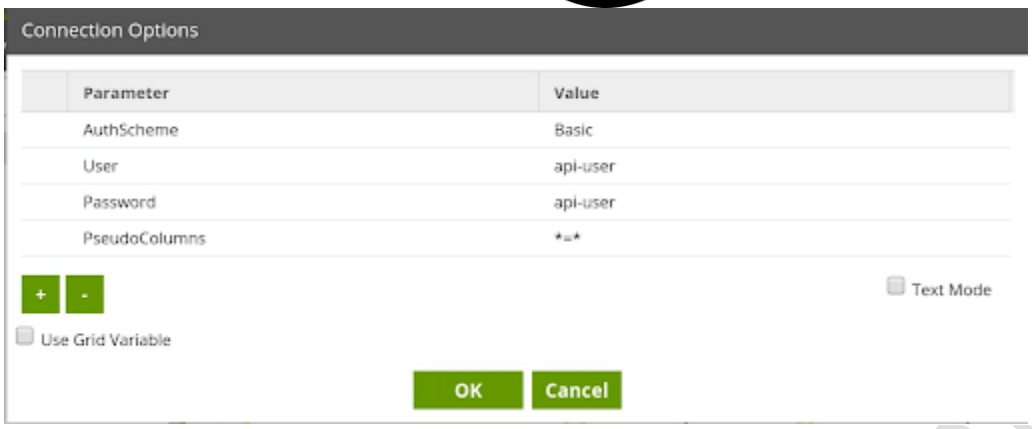
- **AuthScheme**
- **User**
- **Password**

*Note: See our article on **Using KMS Encypted Passwords in Python** for an example of how to securely populate a variable with an encrypted password value. The variable can then be passed to things such as the above mentioned Password parameter.*

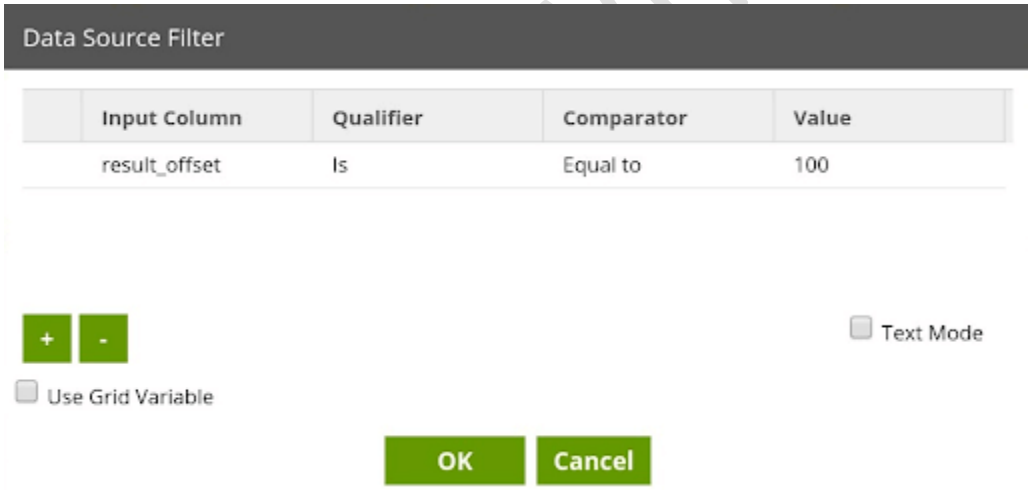
Parameterization

In the article on **Using Parameters with API Profiles,** we demonstrate how to construct a SQL-like query in Advanced Mode to **dynamically pass in a value for an Input parameter defined in an API Profile**. The same thing can be done in Basic Mode by setting an additional Connection Option:

- Parameter: **PseudoColumns**
- Value: *=*



When setting this Connection Option, any Input parameters defined in the referenced API Profile will be made visible in the Data Selection and Data Source Filter properties. To pass in the value of an Input Parameter at runtime, the Data Source Filter property can be used. In the included Matillion ETL jobs, the API Profile for fetching Audit Log data has an “result_offset” Input Parameter defined, which represents the number of audit log records to fetch per page.



Summary

In Part I, we showed how to use the Audit and UserConfig Matillion v1 API endpoints as a source of data to be loaded into a cloud data warehouse. In doing so, we highlighted techniques around implementing common API integration tasks within your Matillion ETL jobs. Next, in Part 2 of our 3 part series on user auditing with Matillion, we will walk through modeling our Audit and UserConfig data into an analytics-ready format and how to populate and

manage Matillion User data in a Type 2 Slowly Changing Dimension in a Matillion ETL job.

References : https://www.matillion.com/blog/security-controls-and-matillion-enable-user-auditing-part-1-of-3