



# Incident Response Platform Integrations

# ODBC Query Function V1.0.0

Release Date: May 2018

Resilient Functions simplify development of integrations by wrapping each activity into an individual workflow component. These components can be easily installed, then used and combined in Resilient workflows. The Resilient platform sends data to the function component that performs an activity then returns the results to the workflow. The results can be acted upon by scripts, rules, and workflow decision points to dynamically orchestrate the security incident response activities.

This guide describes the ODBC Query Function.

Overview

The ODBC Query Function establishes an OBDC connection to the desired SQL database server and executes SELECT, INSERT, UPDATE or DELETE SQL statements.

This package includes a SQL script that creates a sample SQL table with dummy data, example workflows that demonstrate how to call these four SQL statements, rules that start the example workflows and a custom Resilient data table that can be updated by the SELECT statement workflow.

The remainder of this document describes the included function, how to configure it in custom workflows, and any additional customization options.

Installation

Before installing, verify that your environment meets the following prerequisites:

* Resilient platform is version 30 or later.
* You have a Resilient account to use for the integrations. This can be any account that has the permission to view and modify administrator and customization settings, and read and update incidents. You need to know the account username and password.
* You have access to the command line of the Resilient appliance, which hosts the Resilient platform; or to a separate integration server where you will deploy and run the functions code. If using a separate integration server, you must install Python version 2.7.10 or later, or version 3.6 or later, and “pip”. (The Resilient appliance is preconfigured with a suitable version of Python.)

Install the Python components

The functions package contains Python components that will be called by the Resilient platform to execute the functions during your workflows. These components run in the ‘resilient-circuits’ integration framework.

The package also includes Resilient customizations that will be imported into the platform later.

Ensure that the environment is up to date,

sudo pip install --upgrade pip

sudo pip install --upgrade setuptools

sudo pip install --upgrade resilient-circuits

To install the package, you must first unzip it then install the package as follows:

sudo pip install --upgrade fn\_odbc\_query-<*version*>.tar.gz

Configure the Python components

The ‘resilient-circuits’ components run as an unprivileged user, typically named `integration`. If you do not already have an `integration` user configured on your appliance, create it now.

Perform the following to configure and run the integration:

1. Using sudo, become the integration user.

sudo su - integration

1. Use one of the following commands to create or update the resilient-circuits configuration file. Use –c for new environments or –u for existing environments.

resilient-circuits config -c

or

resilient-circuits config -u

1. Edit the resilient-circuits configuration file.
   1. In the [resilient] section, ensure that you provide all the information needed to connect to the Resilient platform.
   2. In the [fn\_odbc\_query] section, edit the settings as follows:

# Define your connection string

sql\_connection\_string=Driver={PostgreSQL};Server=IP Adress;Port=5432;

Database=myDataBase;Uid=myUserName;Pwd=myPassword;

# Optional setting

# Define a list of restricted SQL statements, separated by a comma.

# Example ["delete", "update", "insert"].

# Comment this line if there are no restrictions.

sql\_restricted\_sql\_statements=["delete", "insert", "update"]

# Optional setting

# Define if you wish to execute commits automatically after every SQL statement.

# Comment this line to use false - the default.

sql\_autocommit=true

# Optional setting

# Define a query timeout in seconds.

# Comment this line to use the default 0, which means "no timeout".

sql\_query\_timeout=10

# Optional setting

# Encoding and decoding settings needed for your SQL database.

# Choose one of the supported database types MariaDB, PostgreSQL or MySQL.

# Define which one of supported SQL Server database settings you want to use.

# At the moment MariaDB, PostgreSQL and MySQL are supported.

# Comment this line if you don't wish to configure decoding/encoding.

sql\_database\_type=MariaDB

# Optional setting

# Define number of rows to fetch.

# Comment this line to fetch all.

sql\_number\_of\_records\_returned=10

Connecting to a database

The ODBC function uses an open source Python module, pyodbc, and an ODBC driver to connect to a data source. ODBC drivers are database-specific and are typically written by the manufacturer of the database. More information on pyodbc and ODBC drivers is available on the [GitHub Pyodbc Wiki](https://github.com/mkleehammer/pyodbc/wiki/Drivers-and-Driver-Managers).

Connections to databases are made through the use of connection strings, which are driver-specific. General connection string information for most databases is available at [http://www.connectionstrings.com](http://www.connectionstrings.com/).

Standard PostgreSQL connection string:

Driver={PostgreSQL};Server=IP address;Port=5432;Database=myDataBase;  
Uid=myUsername;Pwd=myPassword;

Standard MySQL connection string:

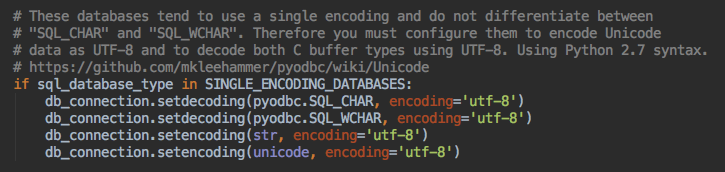
Server=myServerAddress;Port=3306;Database=myDataBase;Uid=myUsername;  
Pwd=myPassword;

Unicode configuration

The pyodbc module recommends configuring ODBC connection's Unicode encoding and decoding settings that are specific for the chosen database and the version of Python in use.

ODBC Function V1.0.0 supports Unicode settings for MariaDB, PostgreSQL and MySQL databases using Python 2.7.

Users may implement additional support by downloading resilient-community-apps and editing the function located in resilient-community-apps/fn\_odbc\_query/fn\_odbc\_query/components/ fn\_odbc\_query.py as shown in the following figure.



More information on Unicode settings is available on [GitHub Pyodbc Wiki](https://github.com/mkleehammer/pyodbc/wiki/Unicode).

Deploy customizations to the Resilient platform

The package contains function definitions that you can use in workflows, and includes example workflows and rules that show how to use these functions.

Deploy these customizations to the Resilient platform with the following command:

resilient-circuits customize

Answer the prompts to deploy functions, message destinations, workflows and rules.

Run the integration framework

To test the integration package before running it in a production environment, you must run the integration manually with the following command:

resilient-circuits run

The resilient-circuits command starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

Configuration of resilient-circuits for restartability

For normal operation, resilient-circuits must run continuously. The recommend way to do this is to configure it to automatically run at startup. On a Red Hat appliance, this is done using a systemd unit file such as the one below. You may need to change the paths to your working directory and app.config.

The unit file should be named ‘resilient\_circuits.service’:

sudo vi /etc/systemd/system/resilient\_circuits.service

The contents:

[Unit]  
Description=Resilient-Circuits Service  
After=resilient.service  
Requires=resilient.service

[Service]  
Type=simple  
User=integration  
WorkingDirectory=/home/integration  
ExecStart=/usr/local/bin/resilient-circuits run  
Restart=always  
TimeoutSec=10  
Environment=APP\_CONFIG\_FILE=/home/integration/.resilient/app.config  
Environment=APP\_LOCK\_FILE=/home/integration/.resilient/resilient\_circuits.lock

[Install]  
WantedBy=multi-user.target

Ensure that the service unit file is correctly permissioned:

sudo chmod 664 /etc/systemd/system/resilient\_circuits.service

Use the systemctl command to manually start, stop, restart and return status on the service:

sudo systemctl resilient\_circuits [start|stop|restart|status]

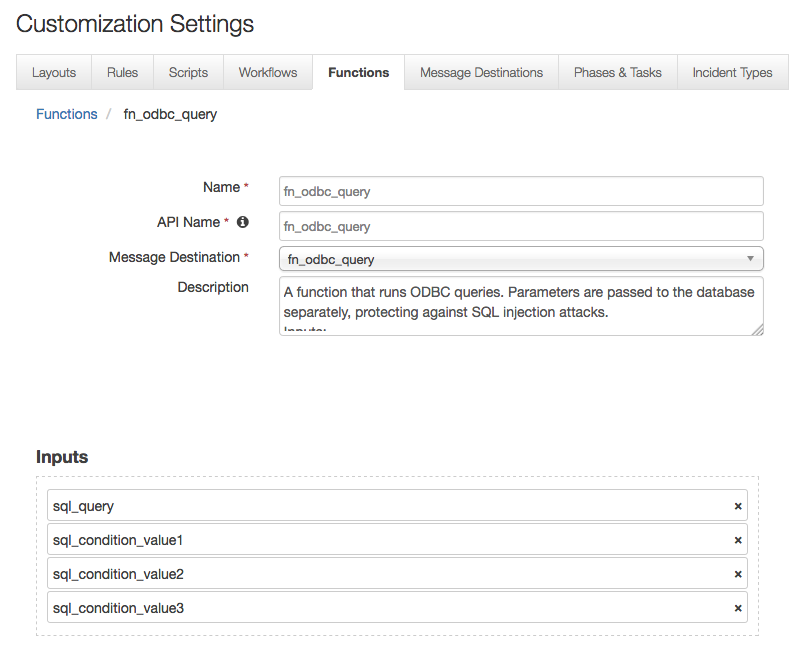
Log files for systemd and the resilient-circuits service can be viewed through the journalctl command:

sudo journalctl -u resilient\_circuits --since "2 hours ago"

Function Description

Once the function package deploys the function, you can view it in the Resilient platform Functions tab, as shown below. The package also includes example workflows and rules that show how the function can be used. You can copy and modify these workflows and rules for your own needs.

In order to try out the included example workflows, users can populate the database with provided SQL script MOCK\_DATA.SQL located in /doc/data directory.

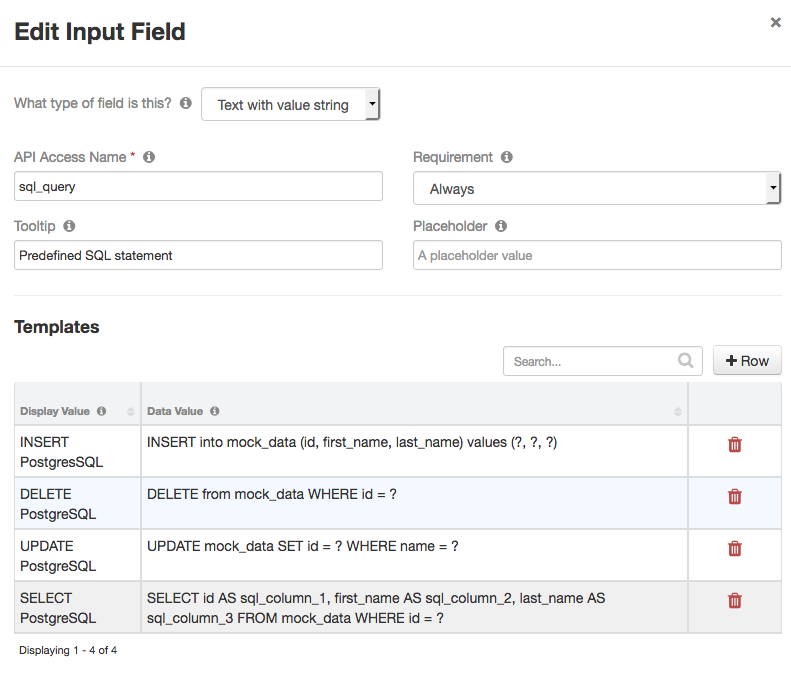


ODBC Query: fn\_odbc\_query

A function that runs ODBC queries. The query and the data/parameters are passed to the database separately, protecting against SQL injection attacks.

This function takes the following input fields:

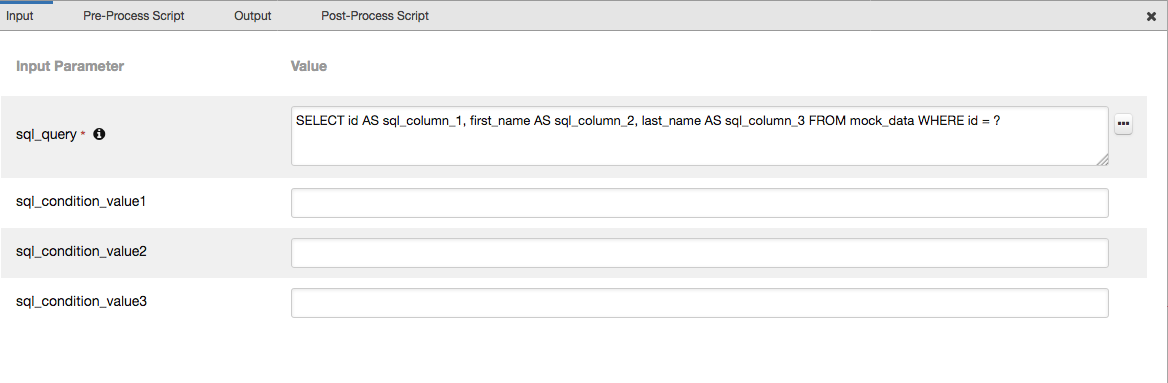
* sql\_query: Query to perform. The field type is “Text with value String” and it contains four predefined example SQL statements that you can choose and edit from within the workflow. The predefined queries, as shown below, contain parameters using a question mark as a placeholder in the SQL query. Users can set values for the question marks in input fields sql\_condition\_value1, sql\_condition\_value2 and sql\_condition\_value3 in the workflow.
* sql\_condition\_value1, sql\_condition\_value2 and sql\_condition\_value3: Parameters used in the query.



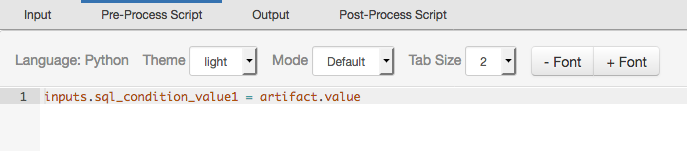
Example ODBC SELECT PostgreSQL Workflow

The “Example ODBC SELECT PostgreSQL” workflow (Object Type = Artifact) calls the ODBC query function. The Input tab of this function is shown in the following figure.

When defining the SQL table column names that the SELECT query will return, it is important to use alias column syntax. Alias names need to match Resilient data table field names. This ensures that query results are saved in the proper Resilient data table field.



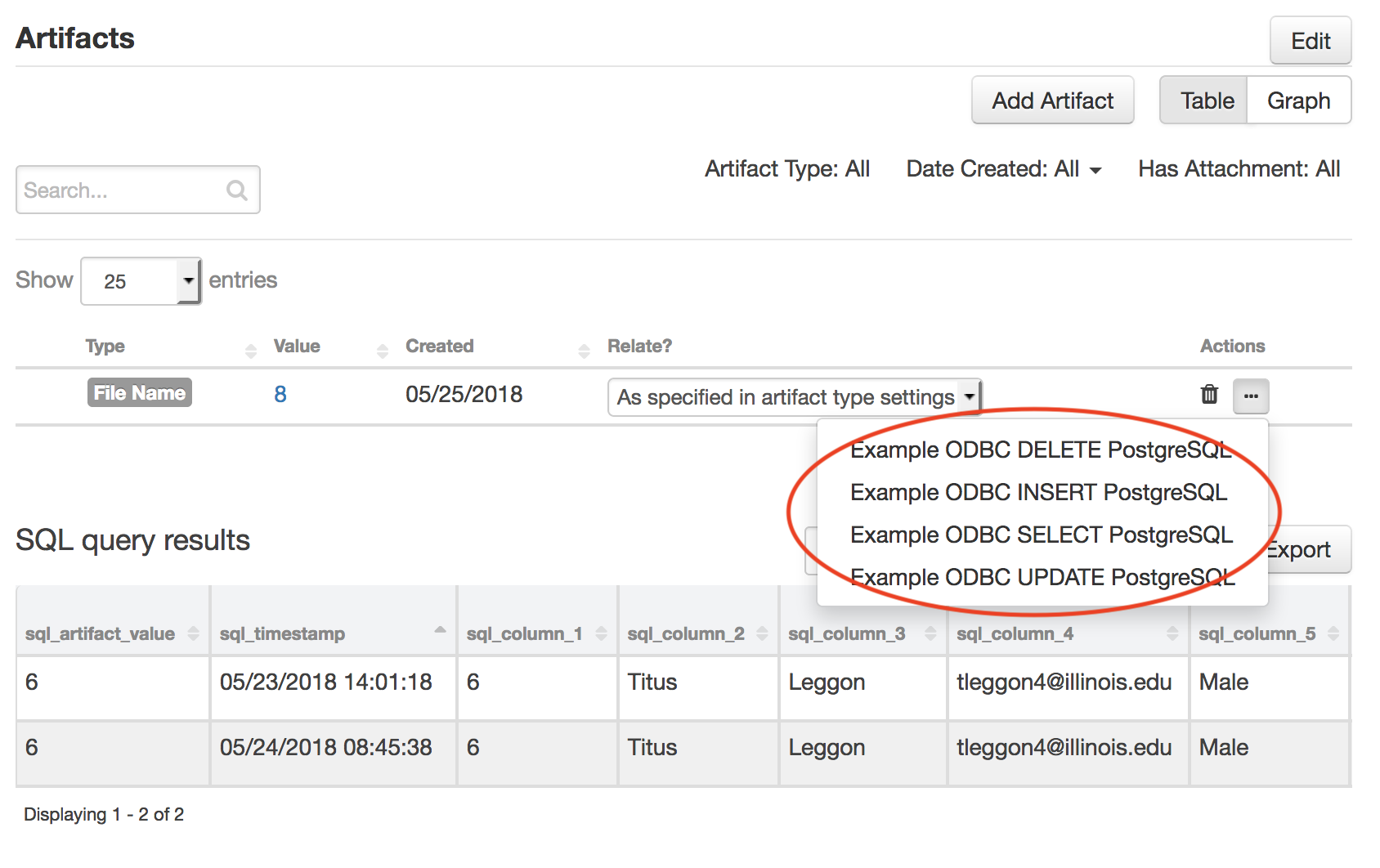
Users may insert data using the sql\_condition\_value1, sql\_condition\_value2 and sql\_condition\_value3 parameters on the Input tab, or set them in the Pre-Process Script to the value of the artifact associated with this workflow as shown in the following figure.



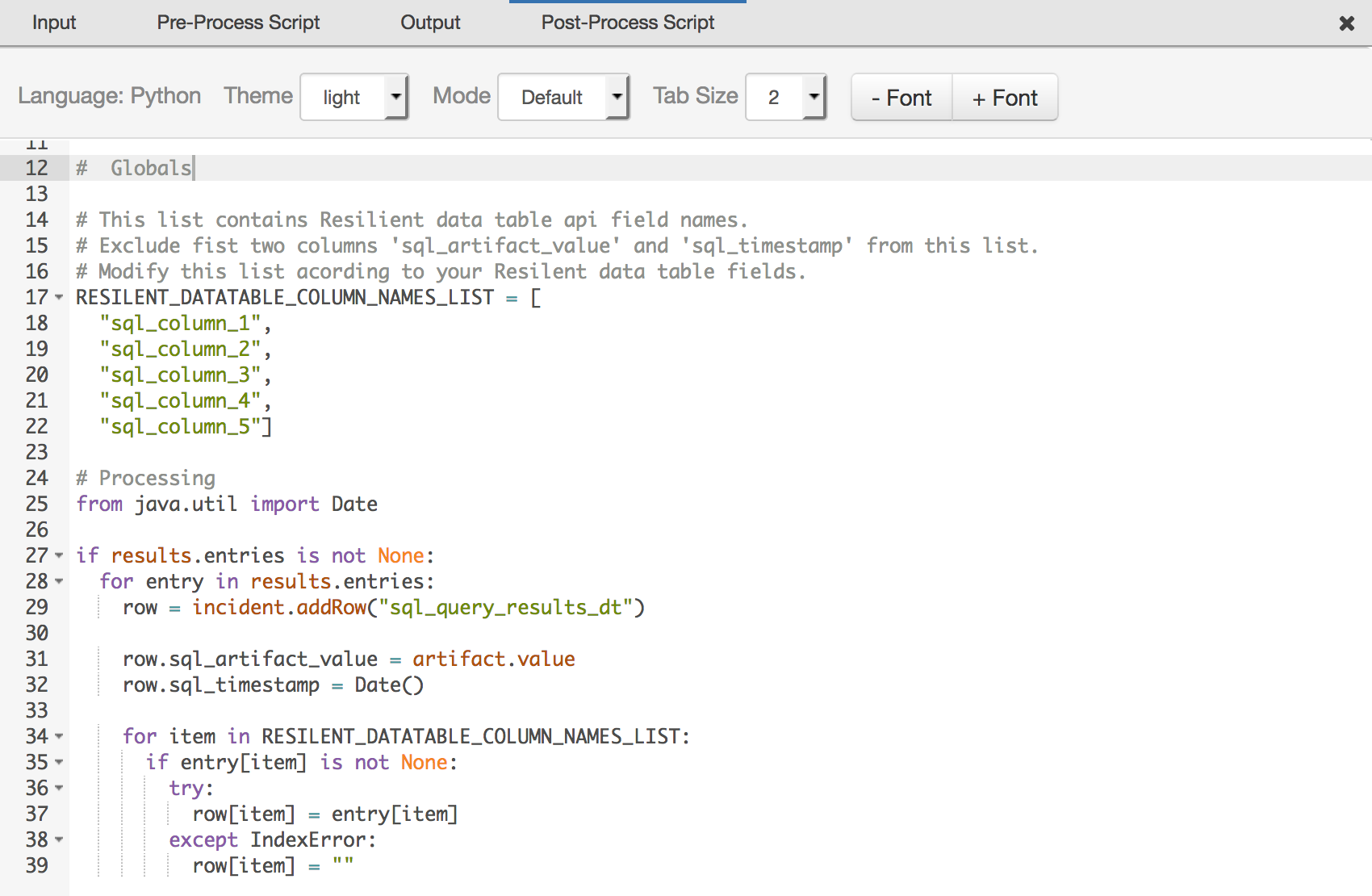
A Menu Item rule called “Example ODBC SELECT PostgreSQL” is also included. This rule calls the provided workflow.

When a user selects this rule from the Actions button on an incident, the rule activates the ODBC Query function. The query results update the custom “sql\_query\_results\_dt” Resilient data table, shown in the following figure. The definition of this data table is also included in the package.

The first two data table fields are set automatically. The ‘sql\_artifact\_value’ field is set to the value of the artifact associated with this workflow. The ‘sql\_timestamp’ field is set to the date and time of the function call. The ‘sql\_column\_1’ to ‘sql\_column\_5’ data table fields are updated by the query results.

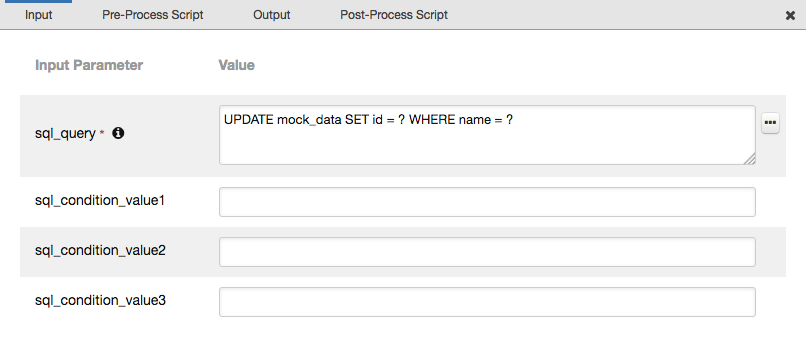


Users may modify the “sql\_query\_results\_dt” data table by adding or removing fields. To ensure SQL query results are saved in proper Resilient data table fields, users need to modify the RESILENT\_DATATABLE\_COLUMN\_NAMES\_LIST in the Post-Process Script as shown in the following figure.

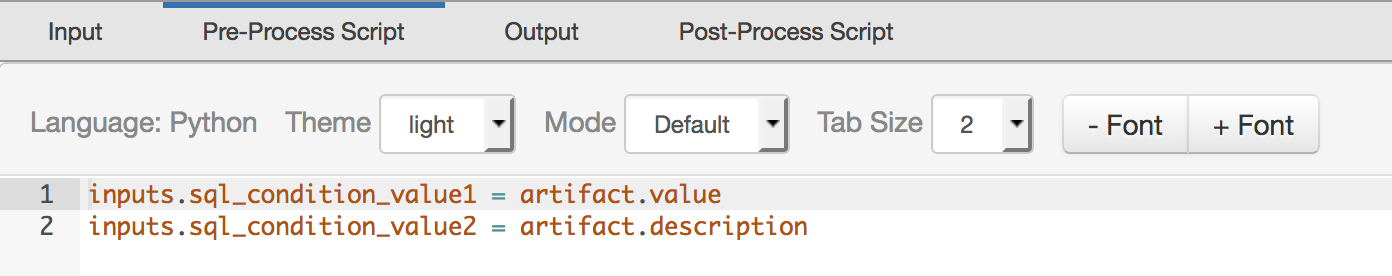


Example ODBC UPDATE PostgreSQL Workflow

The “Example ODBC UPDATE PostgreSQL” workflow (Object Type = Artifact) calls the ODBC query function. The Input tab of this function is shown in the following figure.



Users may insert data using parameters sql\_condition\_value1, sql\_condition\_value2 and sql\_condition\_value3 on the Input tab, or set them in the Pre-Process Script to the value and description of the artifact associated with this workflow as shown in the following figure.

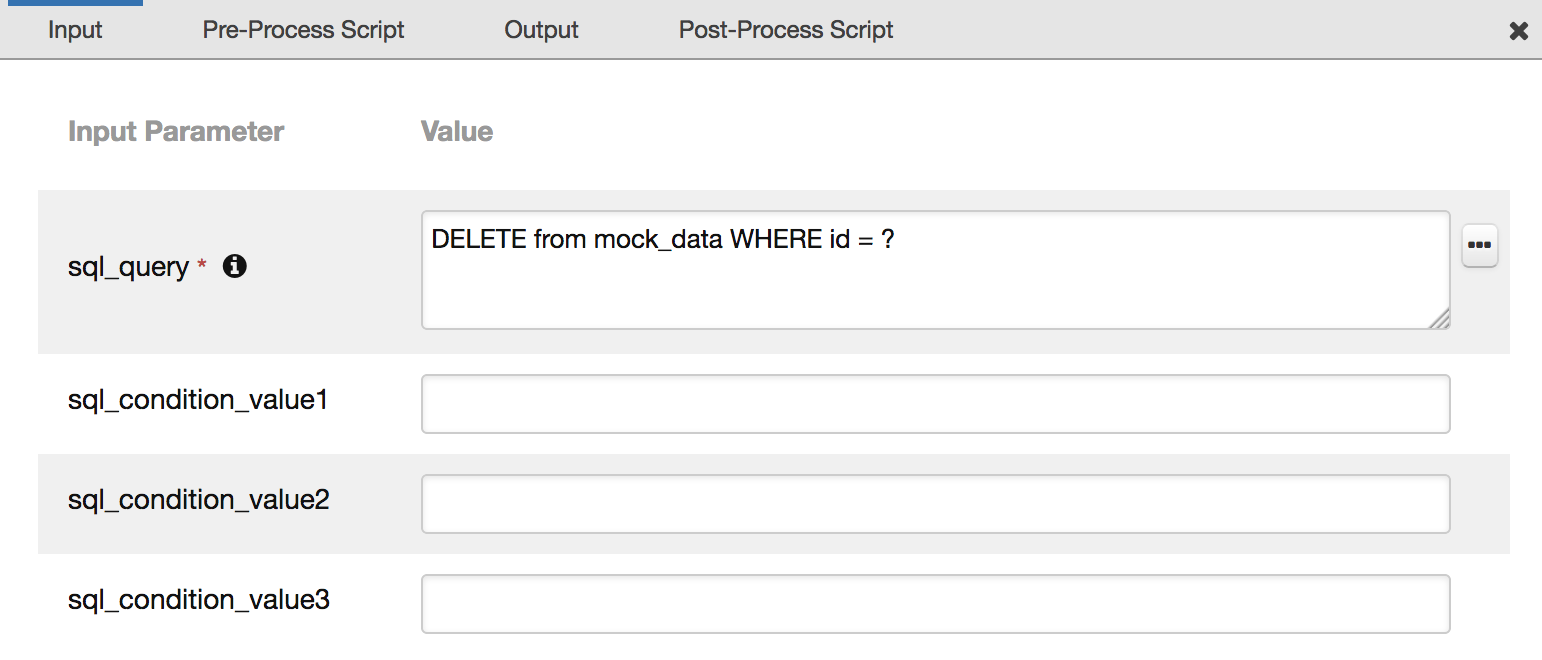


A Menu Item rule called “Example ODBC UPDATE PostgreSQL” is also included. This rule calls the provided workflow.

When a user selects this rule from the Actions button on an incident, the rule activates the ODBC Query function, which then returns the number of processed rows and displays the message in the Action Status.

Example ODBC DELETE PostgreSQL Workflow

The “Example ODBC DELTE PostgreSQL” workflow (Object Type = Artifact) calls the ODBC query function. The Input tab of this function is shown in the following figure.



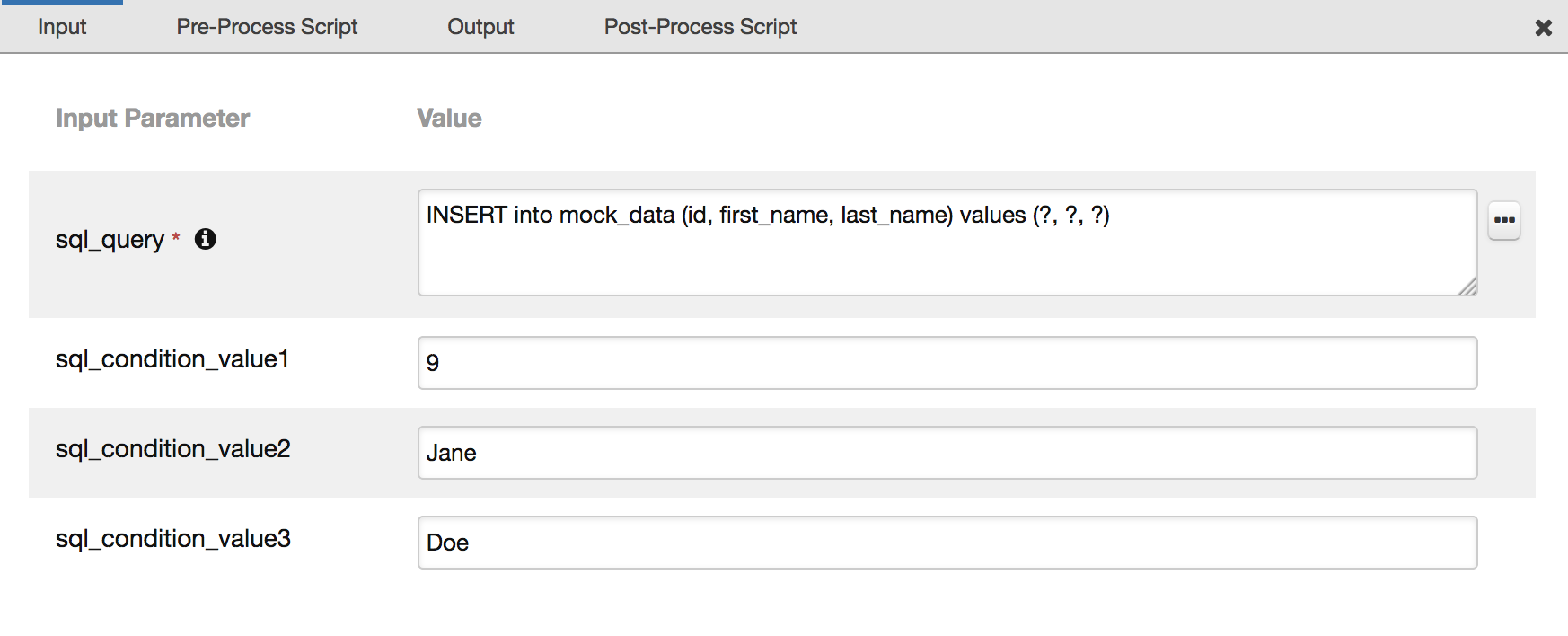
Same as the “Example ODBC SELECT PostgreSQL” workflow, users may insert data using parameter sql\_condition\_value1 on the Input tab, or set it in the Pre-Process Script to the value of the artifact associated with this workflow.

A Menu Item rule called “Example ODBC DELETE PostgreSQL” is also included. This rule calls the provided workflow.

When a user selects this rule from the Actions button on an incident, the rule activates the ODBC Query function, which then returns number of processed rows and displays the message in the Action Status.

Example ODBC INSERT PostgreSQL Workflow

The “Example ODBC INSERT PostgreSQL” workflow (Object Type = Artifact) calls the ODBC query function. The Input tab of this function is shown in the following figure.



Same as the “Example ODBC SELECT PostgreSQL” workflow, users may insert data using parameters sql\_condition\_value1, sql\_condition\_value2 and sql\_condition\_value3 on the Input tab, or set them in the Pre-Process Script to the value of the artifact associated with this workflow.

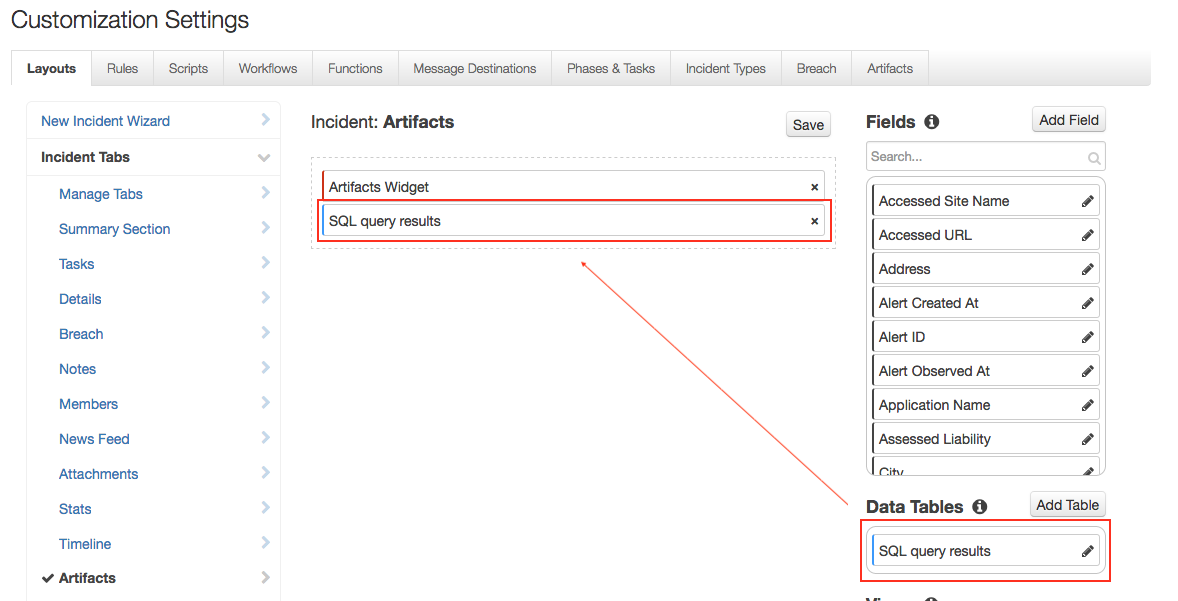
A Menu Item rule called “Example ODBC INSERT PostgreSQL” is also included. This rule calls the provided workflow.

When a user selects this rule from the Actions button on an incident, the rule activates the ODBC Query function, which then returns the number of processed rows and displays the message in the Action Status.

Resilient Platform Configuration

To display query results, users need to manually add the “SQL query results” data table to a new or existing layout.

1. Navigate to the Customization Settings and select or create a new Incident tab in the Layouts tab.
2. Drag the “SQL query results” data table to your Incident tab.
3. Click Save.



Troubleshooting

There are several ways to verify the successful operation of a function.

* Resilient Action Status

When viewing an incident, use the Actions menu to view Action Status. By default, pending and errors are displayed. Modify the filter for actions to also show Completed actions. Clicking on an action displays additional information on the progress made or what error occurred.

* Resilient Scripting Log

A separate log file is available to review scripting errors. This is useful when issues occur in the pre-processing or post-processing scripts. The default location for this log file is: /var/log/resilient-scripting/resilient-scripting.log.

* Resilient Logs

By default, Resilient logs are retained at /usr/share/co3/logs. The client.log may contain additional information regarding the execution of functions.

* Resilient-Circuits

The log is controlled in the .resilient/app.config file under the section [resilient] and the property logdir. The default file name is app.log. Each function will create progress information. Failures will show up as errors and may contain python trace statements.

Support

For additional support, contact [support@resilientsystems.com](mailto:support@resilientsystems.com).

Including relevant information from the log files will help us resolve your issue.