



**Data Feed Integration Guide v1.0**

Incident Response Platform

Licensed Materials – Property of IBM

© Copyright IBM Corp. 2010, 2019.  All Rights Reserved.

US Government Users Restricted Rights: Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Resilient Incident Response Platform Data Feed Integration Guide

|  |  |  |
| --- | --- | --- |
| Version | Publication | Notes |
| 1.0 | April 2019 | Initial release. |

Table of Contents

[1. Overview 5](#_Toc5366942)

[1.1. Architecture 5](#_Toc5366943)

[1.2. Initial Population 6](#_Toc5366944)

[1.3. Useful Tools 6](#_Toc5366945)

[2. Prerequisites 7](#_Toc5366946)

[3. Installation 7](#_Toc5366947)

[3.1. Install the Python components 7](#_Toc5366948)

[3.2. Configure the Python components 7](#_Toc5366949)

[3.3. Deploy customizations to the Resilient platform 8](#_Toc5366950)

[3.4. Run the integration framework 8](#_Toc5366951)

[3.5. Configure Resilient Circuits for restart 9](#_Toc5366952)

[3.6. Confirm deployment 10](#_Toc5366953)

[3.6.1. Configuration of Rules for data tables 10](#_Toc5366954)

[4. Configuration 11](#_Toc5366955)

[4.1. Resilient Circuits config file 11](#_Toc5366956)

[4.2. ODBCFeed class 11](#_Toc5366957)

[4.2.1. Additional connection strings 12](#_Toc5366958)

[4.2.2. Database Field Length Considerations 12](#_Toc5366959)

[4.2.3. Additional considerations 13](#_Toc5366960)

[4.3. FileFeed class 13](#_Toc5366961)

[4.4. SQLiteFeed class 13](#_Toc5366962)

[4.5. ElasticFeed 14](#_Toc5366963)

[4.5.1. Considerations 14](#_Toc5366964)

[4.6. SplunkHECFeed 14](#_Toc5366965)

[4.6.1. Considerations 15](#_Toc5366966)

[5. Test 15](#_Toc5366967)

[6. Preparation Checklist 16](#_Toc5366968)

[7. Configuration and Known Issues 16](#_Toc5366969)

[7.1. All Feeds 16](#_Toc5366970)

[7.2. ODBC Databases 17](#_Toc5366971)

[8. Troubleshoot 18](#_Toc5366972)

[9. Support 18](#_Toc5366973)

[10. Modifications 18](#_Toc5366974)

[10.1. Adding new datastores 18](#_Toc5366975)

[10.2. SQL Dialects 19](#_Toc5366976)

[10.2.1. Modify Dialect Encoding 20](#_Toc5366977)

[10.3. Modify data type mapping 20](#_Toc5366978)

[10.4. Modifying dialect reserved words 21](#_Toc5366979)

[10.5. Modification testing 21](#_Toc5366980)

[11. Appendix: Schemas 21](#_Toc5366981)

# Overview

This guide describes the Resilient Data Feed capability. This functionality allows a Resilient customer to maintain "replica" data from a Resilient platform for access by other tools, such as Business Intelligence solutions. The replications and updates are performed in near real-time. The data written represents the current state of incident data, notes, artifacts, data tables, and so on.

## Architecture

This extension allows you to perform queries on Resilient data without having to access the Resilient database directly. You can then run business intelligence queries from other tools.

The data can be written to any or all of the following destinations:

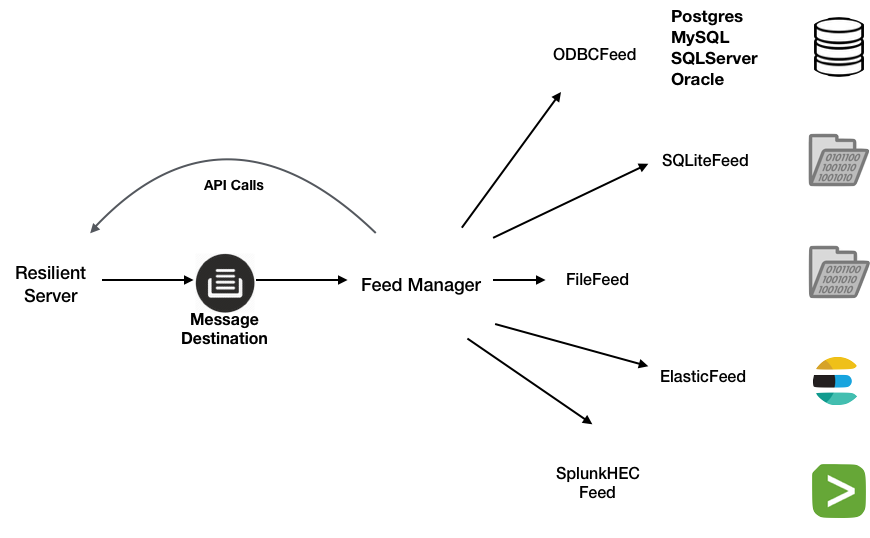
* ODBC databases:
  + PostgreSQL
  + MySQL (MariaDB)
  + Microsoft SQLServer
  + Oracle
* SQLite file/database
* ElasticSearch
* Splunk ES using the HTTP Event Collector
* Local directory (one file per object)

The content is consistent with the Resilient type/field semantic model and includes custom fields. For SQL destinations, the table name is the same as the type name (such as incident, task, and artifact) and the column names are the same as the field "programmatic name".

Data tables are supported (the programmatic name will be the same as the DB table name).

Newly added fields and data tables are dynamically added to the existing destination database.

The architecture allows other destinations to be created with relative ease. The following diagram shows the overall data flow from the Resilient platform to the existing feeds.



## Initial Population

When you run the Data Feed extension against a Resilient platform that has pre-existing data, it can optionally read all that platform’s data and populate your feeds. After the initial data population, the extension performs all updates by listening on a message destination (queue) from which integration logic is called to update the data source.

## Useful Tools

Most "business intelligence" tools allow you to query data from SQL databases. Here are some tools that you can use to run queries and reports against this data:

* Tableau
* Microsoft Power BI
* IBM Cognos
* Grafana
* Interactive SQL, for example:

-- Number of artifacts for each incident

select i.id, i.name, count(\*) num\_artifacts

from incident i join artifact a on i.id = a.inc\_id

group by i.id, i.name;

# Prerequisites

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

* Resilient platform is version 31 or later.
* You have access to a Resilient integration server. An *integration server* is the system that you use to deploy integration packages to the Resilient platform. See the [Resilient Integration Server Guide (PDF)](https://github.com/ibmresilient/resilient-reference/blob/master/developer_guides/Integration%20Server%20Guide.pdf) for more information.
* The following environments have been tested and are recommended:
  + PostgreSQL 9.6 or higher
  + MySQL 5.7 (MariaDB 10.3) or higher
  + Microsoft SQL Server 2017
  + Oracle 12c
  + Splunk ES 7.1
  + ElasticSearch 7.0

# Installation

The integration package contains Python components that are called by the Resilient platform. These components run in the Resilient Circuits integration framework. The package also includes Resilient customizations that will be imported into the platform later.

You perform these installation procedures at the Resilient integration server.

## Install the Python components

Complete the following steps to install the Python components:

1. Ensure that the environment is up-to-date, as follows:

sudo pip install --upgrade pip

sudo pip install --upgrade setuptools

sudo pip install --upgrade resilient-circuits

1. Run the following commands to install the package:

unzip rc-data-feed-1.0.0.tar.gz.zip

sudo pip install --upgrade rc-data-feed-1.0.0.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.

Complete the following steps to configure and run the integration:

1. Using sudo, switch to the integration user, as follows:

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 [-l rc-data-feed]

1. Edit the resilient-circuits configuration file, as follows:
   1. In the [resilient] section, ensure that you provide all the information required to connect to the Resilient platform.
   2. In the [feeds] section, define the feeds you intend to use and create separate sections for each feed. For example:

[feeds]

feed\_names=my\_postgresql\_feed,my\_file\_feed,my\_sqlite\_feed

reload=True

# feed\_data is the default queue that will be listened to

queue=feed\_data

[my\_postgresql\_feed]

class=ODBCFeed

odbc\_connect=Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed

sql\_dialect=PostgreSQL96Dialect

uid=mypguser

pwd=mypassword

[my\_file\_feed]

class=FileFeed

directory=/var/resilient/feed\_data

[my\_sqlite\_feed]

class=SQLiteFeed

file\_name=/var/resilient/feed.sqlite3

See the [Configuration](#_Configuration) section for details.

## Deploy customizations to the Resilient platform

The package contains rules required for the flow of incident data changes to the data feeds.

1. Use the following command to deploy these customizations to the Resilient platform:

resilient-circuits customize [-l rc-data-feed]

1. Respond to the prompts to deploy the message destination 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.

## Configure Resilient Circuits for restart

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 the app.config file.

1. The unit file must be named resilient\_circuits.service To create the file, enter the following command:

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

1. Add the following contents to the file and change as necessary:

[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

1. Ensure that the service unit file is correctly permissioned, as follows:

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

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

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

You can view log files for systemd and the resilient-circuits service using the journalctl command, as follows:

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

## Confirm deployment

Once the package deployment is complete, you can view them in the Resilient platform Rules tab, as shown below.



### Configuration of Rules for data tables

Data tables are also supported by the Data Feeder. Because rules are written to specific data tables, they require a Resilient administrator to create them manually. The following screenshot shows a rule which references a data table and it triggers for any row change (insert, update, delete). Follow this procedure for all the data tables you need to include in the Data Feeder.



# Configuration

There are two aspects to configuration of the Data Feed extension: Resilient Server configuration and app.config configuration.

## Resilient Circuits config file

The configuration goes in the standard app.config file. There is a "feeds" section that contains a "feed\_names" key, which is a comma separated list of feeds that you want to enable. The names you use represent the names of the subsequent sections that contain the actual feed configurations. Each name must be unique. For example:

[feeds]

feed\_names=**my\_postgresql\_feed**,my\_file\_feed,my\_sqlite\_feed

reload=True

# feed\_data is the default queue that will be listened to

queue=feed\_data

[**my\_postgresql\_feed**]

class=ODBCFeed

odbc\_connect=Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed

sql\_dialect=PostgreSQL96Dialect

uid=mypguser

pwd=mypassword

[my\_file\_feed]

class=FileFeed

directory=/var/resilient/feed\_data

[my\_sqlite\_feed]

class=SQLiteFeed

file\_name=/var/resilient/feed.sqlite3

In this example feed\_names references three feeds. The comma separated names refer to additional sections of configuration settings.

This structure allows you to create as many feeds as you need, regardless of the "class". For example, you can have three different ODBC feeds as long as they all have unique names.

You can perform an initial data population by setting reload=True in the app.config [feeds] section and then restarting Resilient Circuits. Be aware that every time Resilient Circuits starts with reload=True, the entire set of Resilient incidents, notes, artifacts, and so on are refreshed in your feeds.

The following sections explain the configuration options for each of the feed classes.

## ODBCFeed class

The ODBCFeed class is probably the most flexible and useful of the feeds. It allows you to write all the incoming data to an ODBC database.

The following configuration items are supported:

| Key | Values | Description |
| --- | --- | --- |
| class | ODBCFeed | Indicates that the section is for an ODBCFeed. |
| odbc\_connect | ODBC connect string | Example for PostgreSQL: Driver={PostgreSQL};Server=localhost;Port=5432;Database=feed |
| sql\_dialect | PostgreSQL96Dialect,  MariaDBDialect,  SQLServerDialect,  OracleDialect | Name of the SQL dialect. |
| uid | DB user name | Specify the database user name in this property and not in the connect string. Most DBs support the uid in the connect string but you should specify in this property instead. |
| pwd | DB password | Specify the database user's password in this property and not in the connect string. Most DBs support the pwd in the connect string but you should specify it in this property instead. You can use the standard Resilient Circuits mechanism for secure password storage. |

When using a data feed database**, IBM Resilient strongly recommends that you create and maintain** the database on system separate from the Resilient platform, where queries cannot impact your running Resilient instance. Allowing access to the Resilient platform for a database instance can also compromise security of the platform itself.

### Additional connection strings

The following table lists additional database connection strings for the other supported databases.

|  |  |
| --- | --- |
| Database | Connection Strings |
| MariaDB | Driver={MariaDB ODBC 3.0 Driver};Server=127.0.0.1;Port=3306; DB=<yourDB>;connectTimeout=0 |
| Oracle | Driver={Oracle 12c ODBC driver};DBQ=ORCLCDB |
| SQLServer | DRIVER={ODBC Driver 17 for SQL Server};SERVER=127.0.0.1;PORT=1443;DATABASE=<yourDB>; |

Your naming of the database drivers (Ex. {MariaDB ODBC 3.0 Driver}) may vary and is specified in your odbcinst.ini file.

Oracle has the further requirement of specifying the connection string references in a TNSNAMES.ORA file. Setting up the Oracle client environment will include the following environment variables (and may include others):

export LD\_LIBRARY\_PATH=/path/to/oracle/libraries/  
export TNS\_ADMIN=/path/to/tnsnames/

### Database Field Length Considerations

Each database type has limits to the size of data stored. The following table describes the limits for each database.

|  |  |  |
| --- | --- | --- |
| Database | Field | Limit |
| Postgres | text | 1GB |
| MySQL/MariaDB | text | 4GB |
| MS SQLServer | varchar(max) | 2GB |
| MS SQLServer | varchar(xx) | 4000 |
| Oracle | nvarchar2(xx) | 2000 |

Some databases support blobs which can be supported specific cases. Blobs are presently unsupported.

### Additional considerations

Some databases have reserve words which cannot be used in tables (such as *date* and *size*). If a Resilient custom field is found to be in a database reserve list, the name (for example, the column name) is altered to include a trailing '\_'.

## FileFeed class

The FileFeed class allows you to write all the incoming data to a directory on your file system (one file per object). The structure of the file names is either:

incident\_{inc\_id}.json (for incident objects)

or (for all other types of objects):

incident\_{inc\_id}\_{type}\_{obj\_id}.json

In this context, "type" can be:

* task
* artifact
* milestone
* note
* data table programmatic name

The following configuration items are supported:

|  |  |  |
| --- | --- | --- |
| Key | Values | Description |
| class | FileFeed | Indicates that the section is for a FileFeed. |
| directory | Path for a directory on local system (required) | Location where the files are to be written. |

## SQLiteFeed class

The SQLiteFeed class allows you to write all the incoming data to a SQLite DB file. SQLite is very useful for testing and in cases where you want to have the data stored in a single file that you can easily share. Some tools may natively support SQLite as well.

SQLite supports CSV formatting, so you can easily export the data from the SQLite file into a CSV file, which can then be imported into another tool, such as Excel, for further analysis.

The following configuration items are supported:

|  |  |  |
| --- | --- | --- |
| Key | Values | Description |
| class | SQLiteFeed | Indicates that the section is for an SQLite. |
| file\_name | Path for a local file on the system where the SQLite DB resides. | This is created if it does not exist. If it does exist, it must be an SQLite database. |

## ElasticFeed

This class allows you to write all incoming data to ElasticSearch. The data representation within Elastic is referenced by index, document type (incident, note, task, artifact, etc.) and document\_id (incident\_id, note\_id, task\_id, etc.).

The following configuration items are supported:

|  |  |  |
| --- | --- | --- |
| Key | Values | Description |
| class | ElasticFeed | Indicates that the section is for an ElasticSearch. |
| url | Ex. https://eliastic.yourorg.com | URL of Elastic server. Port is specified in it’s own parameter. |
| port | Ex. 9200 | Default is 9200 |
| index | Ex. resilient | if using multiple organizations, consider indexes such as resilient<org\_ID> |
| auth\_user  auth\_password |  | User and password to authenticate to ElasticSearch. |
| cafile | True | False | Specify ‘false’ to bypass certification authentication |

### Considerations

ElasticSearch allows for the updating to and deleting of individual documents. No data duplication occurs.

## SplunkHECFeed

The SplunkHECFeed class utilizes the Splunk HTTP Event Collector for data import. This is convenient as the data from Resilient is readily converted to JSON which can be natively consumed by Slunk.

|  |  |  |
| --- | --- | --- |
| Key | Values | Description |
| class | SplunkHECFeed | Indicates that the section is for Splunk ES. |
| token | Ex. 81e2d4bb-c008-49ac-a7a7-c0bf408c999 | API token defined for the HEC data input. |
| host | Ex. splunk.yourorg.com |  |
| port | Ex. 8088 | The default is 8088 |
| index | Ex data\_feeder | The name of the index already defined. |
| event\_host | Ex. myorg.com | Optional host name to record as the source of the events. |
| event\_source=resilient | Ex. resilient | Optional source name of the events. Specifying a value improves searching |
| event\_source\_type |  | Optional source\_type if one value is used for all events. If unspecified, each object type (incident, task, note, etc.) is used as the source\_type |
| use\_ssl | True | False | Indicate if connections to the HEC uses encryption (https) |

### Considerations

Enable the HTTP Event Collector within Splunk ES before using this data feed.

Splunk events are immutable. Resilient object changes are represented as new events. No event deletion is possible.

# Test

A simple test can be performed by setting up a data feed with the feed\_names=file\_feed datastore and the parameter reload=False. Once configured and resilient-circuits is running, perform an update to an incident and confirm that the resilient-circuits logs contain an update

2019-04-04 21:16:02,165 INFO [file\_feed] Inserting/updating incident; id = 2783460

The resulting file (incident\_2783460.json) will contain a json formatted representation of the incident data similar to the following

{

"addr": null,

"alberta\_health\_risk\_assessment": null,

"hard\_liability": 0,

"city": null,

"country": "United States",

"creator\_id": "able baker (a@example.com)",

"crimestatus\_id": "Unknown",

"data\_encrypted": null,

"data\_format": null,

"end\_date": "2019-03-14T13:06:48.550000",

"create\_date": "2019-02-13T18:38:55",

"discovered\_date": "2019-02-13T18:38:40",

"start\_date": null,

"exposure\_dept\_id": null,

"description": "<div class=\"rte\"><div>new description new again -- updated</div></div>",

"employee\_involved": null,

…

}

Now repeat the same test with your production datastore(s). Data is not synchronized until new Resilient objects are created or existing ones are updated.

# Preparation Checklist

In order to successfully use this extension, ensure you have built out your environments outlined in the sections above, and reviewed the requirements below:

1. Identified all the objects you will perform analytics on, including data tables. Data tables will require additional rules added for synchronization to your datastore.
2. Configured all aspects of your datastore
   1. For odbc datastores, install and test the appropriate drivers and configuration settings.
   2. Build out the database to use and the account has the correct permissions for data access.
3. Added the datastore configuration data to your app.config file, including the feed(s) to use.
4. Confirmed that your BI tool has access to the datastore environment.
5. Reviewed your use of the reload app.config setting as this will have potential performance consequences when resilient-circuits is restarted.
6. Completed the tests defined in section 5.

# Configuration and Known Issues

There are a number of issues to be aware of when using the Data Feed

All Feeds

* When deleting a Resilient incident, that record will be deleted from the replica datastore. However, all the incident’s associated tasks, notes, artifacts, datatables, etc. are not removed.
* For sizing purposes, here are some guidelines for the size of each object type. Use this information for estimation of your file system and database requirements.

|  |  |
| --- | --- |
| Incident | ~ 4000 characters |
| Tasks | 2000-5000 characters |
| Notes | 1000-3000 characters |
| Artifacts | < 500 characters |
| Attachments | Up to 20mb |
| Data tables | 200-2000 characters per row |

* Incident HIPAA fields when directly edited are not synchronized in near-time. Any additional change to an incident will then also update the HIPAA fields.

ODBC Databases

* All databases require their own set of drivers to be installed. This package does not install the libraries necessary as those operations are specific to the database used, Once the necessary driver(s) and installed, this information is captured in the odbcinst.ini file (for instance, found here: /usr/local/etc/odbcinst.inifile) similar to the following:

[PostresSQL Driver]

Description=ODBC for PostgreSQL

Driver=/usr/local/lib/psqlodbcw.so

[MariaDB ODBC 3.0 Driver]

Description=MariaDB Connector/ODBC v.3.0

Driver=/usr/local/Cellar/mariadb-connector-odbc/3.0.2/lib/libmaodbc.dylib

[ODBC Driver 17 for SQL Server]

Description=Microsoft ODBC Driver 17 for SQL Server

Driver=/usr/local/lib/libmsodbcsql.17.dylib

UsageCount=1

[Oracle 12c ODBC driver]

Description = Oracle ODBC driver for Oracle 12c

Driver = /usr/local/lib/libsqora.so.12.1

Setup =

FileUsage =

CPTimeout =

CPReuse =

* All data in Resilient is encoded as UTF-8. When creating the initial database, ensure that a similar encoding is specified.
* When creating the database user account which will access the defined database, provide the necessary permissions to allow full access to create tables, alter tables by adding columns, and full capability to insert, update and delete records.   
  These are the permissions granted when creating a user to reference a SQLServer database:

**CREATE** **LOGIN** res\_db **WITH** PASSWORD = '\*\*\*\*\*’;

**CREATE USER** res\_db **FOR** **LOGIN** res\_db;

**GRANT** **UPDATE,INSERT,DELETE,SELECT,ALTER,CREATE TABLE TO** res\_db;

* Deleting a custom field in Resilient will not remove it from the Data Feed datastore.
* A custom field changed between Text and TextArea will have no effect in the Data Feed process.
* Python 2.7 with pyodbc 4.0.X will lose the millisecond precision on datetime fields. If this value is important to your environment, use Python 3.6.X or greater.
* A custom field deleted in Resilient and recreated as a different data type (for instance, a text field recreated as number) will break the feed process. It’s recommended one also delete the existing datastore table column so that it can be automatically recreated when resilient-circuits is restarted.
* No foreign keys are created between tables. However, each table contains the inc\_id column which can be used to link tables together. Below is a sample SQL query linking an incident with its tasks.

**SELECT** \* **FROM** incident   
**INNER** **JOIN** task  
**ON** incident.inc\_id = task.inc\_id  
**WHERE** incident.inc\_id=2095;

# Troubleshoot

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

* 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.

# Modifications

Several code modifications are possible. This section gives you the necessary hints for how to consider making changes. In all cases, make sure you have a backup of the original code.

## Adding new datastores

If you’re interested in adding a completely new datastore, start by adding its name to the list defined in components/feed\_ingest.py and adding an import statement for the Destination class

from rc\_data\_feed.lib.feed import FeedContext

**from rc\_data\_feed.lib.my\_feed import MyFeedDestination**

**…**  
  
**class** FeedComponent(ResilientComponent):  
 *"""This component handles initial population of a feed and ongoing  
 modifications from the associated queue."""* DATATABLE\_TYPE\_ID = 8  
 INCIDENT\_TYPE\_ID = 0  
 INC\_PAGE\_SIZE = 500  
 SEARCH\_PAGE\_SIZE = 50  
 AVAILABLE\_CLASSES = {  
 "ODBCFeed": ODBCFeedDestination,  
 "FileFeed": FileFeedDestination,  
 "ElasticFeed": ElasticFeedDestination,  
 "SQLiteFeed": SqliteFeedDestination,  
 "SplunkHECFeed": SplunkHECFeedDestination**,**

**"MyFeed": MyFeedDestination**  
 }

Your new class will reside in the lib/ directory and the class will inherent from FeedDestinationBase class.

Each new class will override the FeedDestinationBase send\_data() function. This function defines the steps needed to convert the Resilient json data into the format necessary for your datastore. The helper routine context.type\_info.flatten() can be used to flatten the Resilient json data to a simple key/value pair json representation which many datastores can natively consume.

The send\_data() function can further support delete logic as well as insert/update logic by testing the boolean variable context.is\_deleted.

## SQL Dialects

Each SQL database dialect is defined in lib/sql\_dialect.py. Here you’ll find the existing dialect classes such as SqliteDialect, PostgresSQL96Dialect, MySqlDialect, SqlServerDialect and OracleDialect. Each of these dialects are referenced in your app.config file using the sql\_dialect= parameter

[postgres\_feed]  
class=ODBCFeed  
odbc\_connect=Driver={PostresSQL Driver};Server=127.0.0.1;DB=res\_test;Port=5432;connectTimeout=0  
**sql\_dialect=PostgreSQL96Dialect**  
uid=res\_test  
pwd=res\_test

To add a new dialect, specify a new class inherited from the ODBCDialectBase base class.

Each dialect needs to define the logic for the operations below. These operations are defined in the SqlDialect base class. These functions build the SQL statements executed on by the standard sequence logic defined in components/feed\_ingest.\_populate\_incidents() and \_populate\_others().

get\_upsert()

get\_delete()

get\_create\_table\_if\_not\_exists()

get\_add\_column\_to\_table()

There are additional functions specified in ODBCDialectBase which may be overwritten:

get\_parameters()

configure\_connection()

get\_column\_type()

The best of advice is to copy an existing dialect and modify the existing functions to match the requirements of your database.

### Modify Dialect Encoding

Each SQL dialect modifies the database connection with the encoding used to read and write data. The configure\_connection() function makes these changes and in all cases, sets the encoding to UTF-8. In some cases, the format of the encoding parameters may change between python environments and the following logic can be used to account for those different environments:

**def** configure\_connection(self, connection):  
 connection.setdecoding(pyodbc.SQL\_WCHAR, encoding=**'utf-8'**) *# pylint: disable=c-extension-no-member* **if** sys.version\_info.major == 2: *# to set encoding on python 2* connection.setencoding(str, encoding=**'utf-8'**)  
 connection.setencoding(unicode, encoding=**'utf-8'**)  
 **else**: *# an issue and try encoding without specifying fromtype* connection.setencoding(encoding=**'utf-8'**)

### Modify data type mapping

Each SQL dialect contains a mapping table to convert Resilient data types to your dialect’s datastore. See the function get\_column\_type() for how mapping is presently done.

**def** get\_column\_type(self, input\_type): # pylint: disable=no-self-use  
 """  
 Gets the DB column type for the specified Resilient 'input type'  
  
 **:**param input\_type: The Resilient input type value (e.g. datepicker,   
 boolean, number, text, text\_area, etc.)  
  
 **:**returns The DB type to use for this dialect.  
 """  
 type\_dict = dict(  
 number=**'BIGINT'**,  
 datepicker=**'DATE'**,  
 datetimepicker=**'TIMESTAMP'**,  
 boolean=**'BOOLEAN'** )  
  
 **if** input\_type **in** type\_dict:  
 **return** type\_dict[input\_type]  
  
 **return 'TEXT'**

Text data types can be challenging for some databases based on the data length limits. In those cases, use a constant to define the limit and truncate your data using that value. Be aware that character limits do not account for Unicode characters which are double-byte. This means that a data limit of 64k characters actually translates to 32k Unicode characters.

MAX\_MARIADB\_TEXT = 32000

**def** get\_parameters(self, parameter\_names, parameters):  
 *# Need to get a list that contains all the values in the same order as parameter\_names.* bind\_parameters = list()  
  
 **for** name **in** parameter\_names:  
 bind\_parameters.append(parameters[name][:MAX\_MARIADB\_TEXT] **if** isinstance(parameters[name], string\_types) **else** parameters[name])

### Modifying dialect reserved words

Each SQL Dialect contains reserved words which cannot be used in database table and column definitions. To ensure all data fields in Resilient can be stored, a list of reserve words is maintained. When a conflict is found the resulting table or column name has an underscore added to the name. From time to time, new database releases add to this reserved word list. Just add those words to an existing list:

RESERVE\_LIST = ['all', 'analyse', 'analyze', 'and', 'any', 'array', 'as',   
 'asc', 'asymmetric', 'both', 'case', 'cast',  
 'check', 'collate', 'column', 'constraint', 'create',   
 'current\_date', 'current\_role', 'current\_time',  
 'current\_timestamp', 'current\_user', 'default',

…  
 ]

## Datastore testing

When adding any new datastore or modifying an existing one, it’s good practice to create or exercise test cases to ensure the data is properly stored. See the existing tests in the tests/ folder for ways to ensure all data types are stored and retrieved correctly. These tests are run using pytest:

pytest -v tests/test\_postressql.py

**================================================================   
test session starts =================================================================**

**collected 7 items**

tests/test\_postressql.py::test\_get\_parameters PASSED

tests/test\_postressql.py::test\_createtable PASSED

tests/test\_postressql.py::test\_insert\_row PASSED

tests/test\_postressql.py::test\_update\_row PASSED

tests/test\_postressql.py::test\_delete\_row PASSED

tests/test\_postressql.py::test\_altertable PASSED

tests/test\_postressql.py::test\_droptable PASSED

# Appendix: Schemas

This sample represents the schema for a Postgres DB built-in objects (incident, artifact, attachment, note, milestone and task). Your schema will vary when custom fields are added as well as datatables added. This schema presents a complete representation of the fields stored within Resilient.

This list was generated via the following sql command

SELECT columns.table\_name, column\_name, data\_type FROM information\_schema.tables   
inner join INFORMATION\_SCHEMA.COLUMNS  
on columns.table\_name = tables.table\_name  
where tables.table\_catalog='res\_test' and tables.table\_schema='public'  
and tables.table\_name in ('incident', 'artifact', 'note', 'task', 'milestone', 'attachment')  
order by columns.table\_name, column\_name;

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| artifact | description | text |
| artifact | hits | text |
| artifact | id | integer |
| artifact | inc\_id | integer |
| artifact | relating | boolean |
| artifact | type | text |
| artifact | value | text |

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| attachment | content\_type | text |
| attachment | created | timestamp without time zone |
| attachment | creator\_id | text |
| attachment | id | integer |
| attachment | inc\_id | integer |
| attachment | name | text |
| attachment | size | bigint |

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| incident | addr | text |
| incident | alberta\_health\_risk\_assessment | boolean |
| incident | city | text |
| incident | confirmed | boolean |
| incident | country | text |
| incident | create\_date | timestamp without time zone |
| incident | creator\_id | text |
| incident | crimestatus\_id | text |
| incident | data\_compromised | boolean |
| incident | data\_contained | boolean |
| incident | data\_encrypted | boolean |
| incident | data\_format | text |
| incident | data\_source\_ids | text |
| incident | description | text |
| incident | discovered\_date | timestamp without time zone |
| incident | due\_date | timestamp without time zone |
| incident | employee\_involved | boolean |
| incident | end\_date | timestamp without time zone |
| incident | exposure\_dept\_id | text |
| incident | exposure\_individual\_name | text |
| incident | exposure\_type\_id | text |
| incident | exposure\_vendor\_id | text |
| incident | gdpr\_breach\_circumstances | text |
| incident | gdpr\_breach\_type | text |
| incident | gdpr\_breach\_type\_comment | text |
| incident | gdpr\_consequences | text |
| incident | gdpr\_consequences\_comment | text |
| incident | gdpr\_final\_assessment | text |
| incident | gdpr\_final\_assessment\_comment | text |
| incident | gdpr\_harm\_risk | text |
| incident | gdpr\_identification | text |
| incident | gdpr\_identification\_comment | text |
| incident | gdpr\_lawful\_data\_processing\_categories | text |
| incident | gdpr\_personal\_data | text |
| incident | gdpr\_personal\_data\_comment | text |
| incident | gdpr\_subsequent\_notification | boolean |
| incident | hard\_liability | bigint |
| incident | harmstatus\_id | text |
| incident | hipaa\_acquired\_comment | text |
| incident | hipaa\_additional\_misuse\_comment | text |
| incident | hipaa\_additional\_misuse | boolean |
| incident | hipaa\_adverse\_comment | text |
| incident | hipaa\_breach\_comment | text |
| incident | hipaa\_breach | boolean |
| incident | hipaa\_adverse | boolean |
| incident | hipaa\_acquired | boolean |
| incident | hipaa\_misused\_comment | text |
| incident | hipaa\_misused | boolean |
| incident | id | integer |
| incident | impact\_likely | boolean |
| incident | inc\_id | integer |
| incident | inc\_training | boolean |
| incident | incident\_type\_ids | text |
| incident | jurisdiction\_name | text |
| incident | members | text |
| incident | name | text |
| incident | negative\_pr\_likely | boolean |
| incident | nist\_attack\_vectors | text |
| incident | owner\_id | text |
| incident | org\_id | number |
| incident | phase\_id | text |
| incident | plan\_status | text |
| incident | reporter | text |
| incident | resolution\_id | text |
| incident | resolution\_summary | text |
| incident | severity\_code | text |
| incident | start\_date | timestamp without time zone |
| incident | state | text |
| incident | workspace | text |
| incident | zip | text |

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| milestone | date | timestamp without time zone |
| milestone | description | text |
| milestone | id | integer |
| milestone | inc\_id | integer |
| milestone | title | text |

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| note | create\_date | timestamp without time zone |
| note | id | integer |
| note | inc\_id | integer |
| note | mentioned\_users | text |
| note | text | text |
| note | user\_id | text |

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Data Type |
| task | active | boolean |
| task | at\_id | bigint |
| task | attachments\_count | bigint |
| task | cat\_name | text |
| task | category\_id | text |
| task | closed\_date | timestamp without time zone |
| task | custom | boolean |
| task | description | text |
| task | due\_date | timestamp without time zone |
| task | id | integer |
| task | inc\_id | integer |
| task | inc\_name | text |
| task | init\_date | timestamp without time zone |
| task | instr\_text | text |
| task | instructions | text |
| task | members | text |
| task | name | text |
| task | notes\_count | bigint |
| task | owner\_id | text |
| task | phase\_id | text |
| task | private | boolean |
| task | required | boolean |
| task | status | text |