



# Incident Response Platform Integrations

# Slack Function V1.0.2

Release Date: August 2020

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, and returns the results to the workflow. The results can then be used by scripts, rules, and workflow decision points to dynamically orchestrate the security incident response activities.

This guide describes the Slack Integration.

Overview

Slack is an online communications solution allowing communities to communicate as groups or directly with each other through conversations and video conferences in Slack channels. This Resilient platform functions-based integration enables Incident, Note, Artifact, Task, and Attachment data to be shared in Slack. Users can create or designate private or public channels in a Slack workspace and invite Slack users to the channels. Users can also customize data from the Resilient objects to post in a channel, as well as archive channels.

There are three functions and eight example workflows in this integration package. This document describes the included functions, how to configure them in custom workflows, and demonstrates additional customization options.

History

|  |  |
| --- | --- |
| Version | Comment |
| 1.0.2 | Support for App Host,  Proxy support added |

App Host Installation

All the components for running Slack in a container already exist when using the App Host app. The remainder of this section details the Slack configuration file changes.  
  
It's possible to override the template used for archiving a channel.   
Use the app.config setting: template\_file=/var/rescircuits/slack\_template.jinja2to reference the template file named slack\_template.jinja2at location /var/rescircuits. See the Template file section below for the default jinaj2 code.

Integration Server Installation

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

* Resilient platform is version 35 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 a Resilient integration server where you will deploy and run the functions code. If not, you need to install and configure the server as described in the [Integration Server Guide](https://www.ibm.com/support/knowledgecenter/SSBRUQ_37.0.0/doc/container_apps.html).

Slack configuration

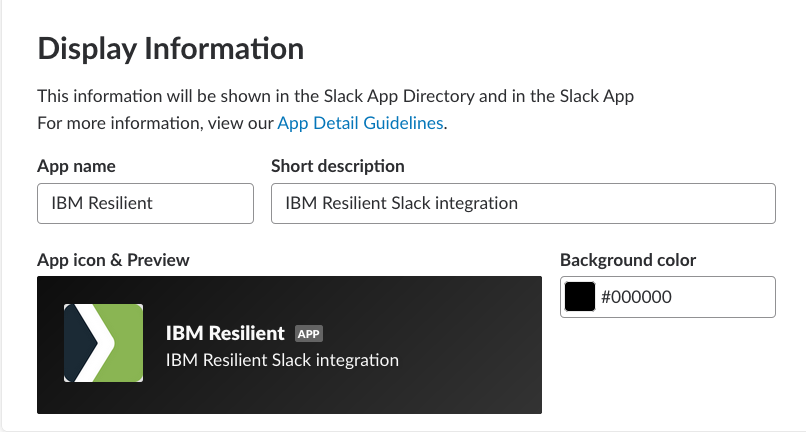
Prior to installing the Slack function, follow the Slack documentation (<https://api.slack.com/slack-apps>) to build a new Slack App.

In the *Basic Information* page, find the *Display Information* section as shown below, then configure your App name and upload the App icon. You may use the Resilient logo for your Slack App icon. ResilientLogo.png is included in the package and can be found in the fn\_slack/doc directory. Make sure to save changes.

Your App’s name is used for message authorship. How message authorship is attributed depends on a few factors, with some behaviors varying based on the kinds of tokens being used to post a message.

The App’s name and icon is used as the author of the posted messages. Users can change message authorship in the app.config file or example workflows.

When uploading files, the name of the authenticated user of the Slack App is used for authorship. This behavior can be changed by adding a [Bot User](https://api.slack.com/bot-users).



The Resilient integration requires that certain permissions are enabled. After you create a Slack App and set its Display Information, navigate to *OAuth & Permissions* page. Scroll down to the *Scopes* section and add the following permission scopes:

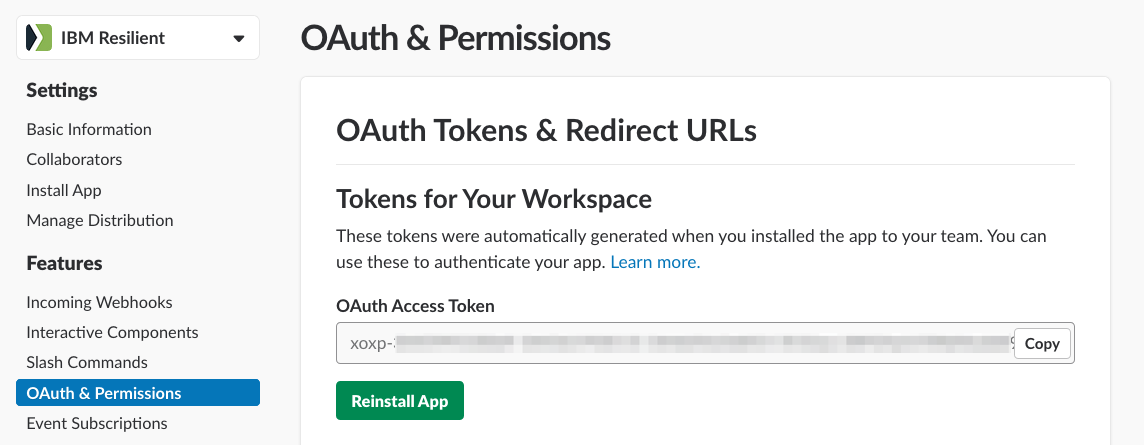
* channels:history -- to access user’s public channels
* channels:read -- to access information about user’s public channels
* channels:write -- to modify your public channels
* chat:write:bot -- to send messages as Slack App name
* chat:write:user -- to send messages as user
* groups:history -- to access content in user’s private channels
* groups:read -- to access information about user’s private channels

**NOTE**: The function only retrieves those private channels from your Slack workspace in which the Slack App’s authorized user has been invited.

* groups:write -- to modify your private channels
* files:write:user – to upload and modify files as user
* users:read – to access your workspace’s profile information
* users:read.email -- to view email addresses of people on this workspace

Make sure to save your changes and then either click on **Install App to Workspace** or **Reinstall App** button in the *OAuth Tokens & Redirect URLs* section and authorize changes in the next summary page.

Installation generates an **OAuth Access Token** as shown below.



Install the Python components

The slack 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:

unzip app-fn\_slack-x.x.x.zip

sudo pip install fn\_slack-x.x.x.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\_slack] section, edit the settings as follows:

# Slack app OAuth Access Token

api\_token=xoxp-xxxxxxxxx-xxxxxxxxxxxx-xxxxxxxxxxxxx-xxxxxxxxxxx

# Username represents the default submission author.

# Used together with 'as\_user=False'.

# You can also update the username on the Workflow.

username=Resilient

# template file override  
#template\_file=/var/rescircuits/slack\_template.jinja2  
  
# add proxy support here or use [integrations] for integration wide proxy settings  
#http\_proxy=  
#https\_proxy=

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.

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

resilient-circuits customize

1. Respond to 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.

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 app.config.

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"

Test the Integration

Many integrations come with a self-test capability. This feature is enabled in the integration and with resilient-circuits version 30.0.111 or greater. Once the integration is configured in the resilient-circuits configuration file, testing to the end-point solution can be performed with the following command:

resilient-circuits selftest [-l fn-slack]

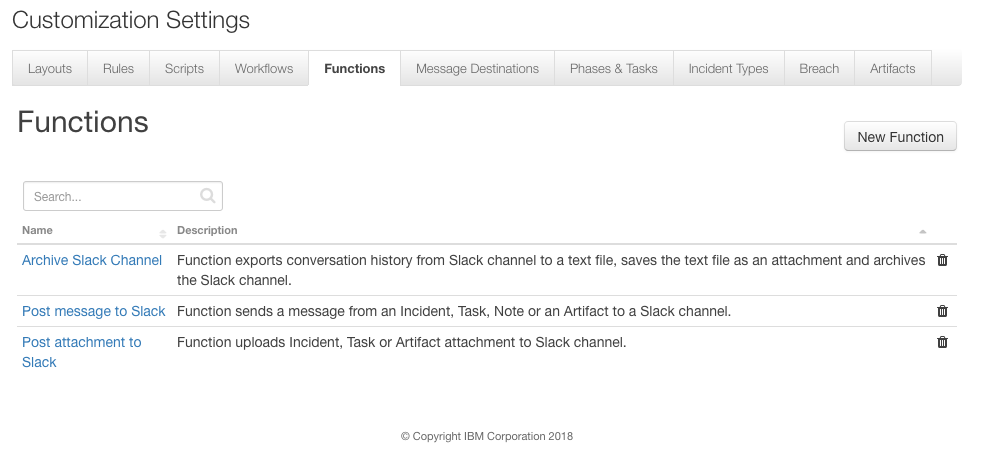
The resulting file will produce a result indicating success, failure or a message indicating that the feature is not available. Here are a few examples:

resilient-circuits selftest -l fn-slack

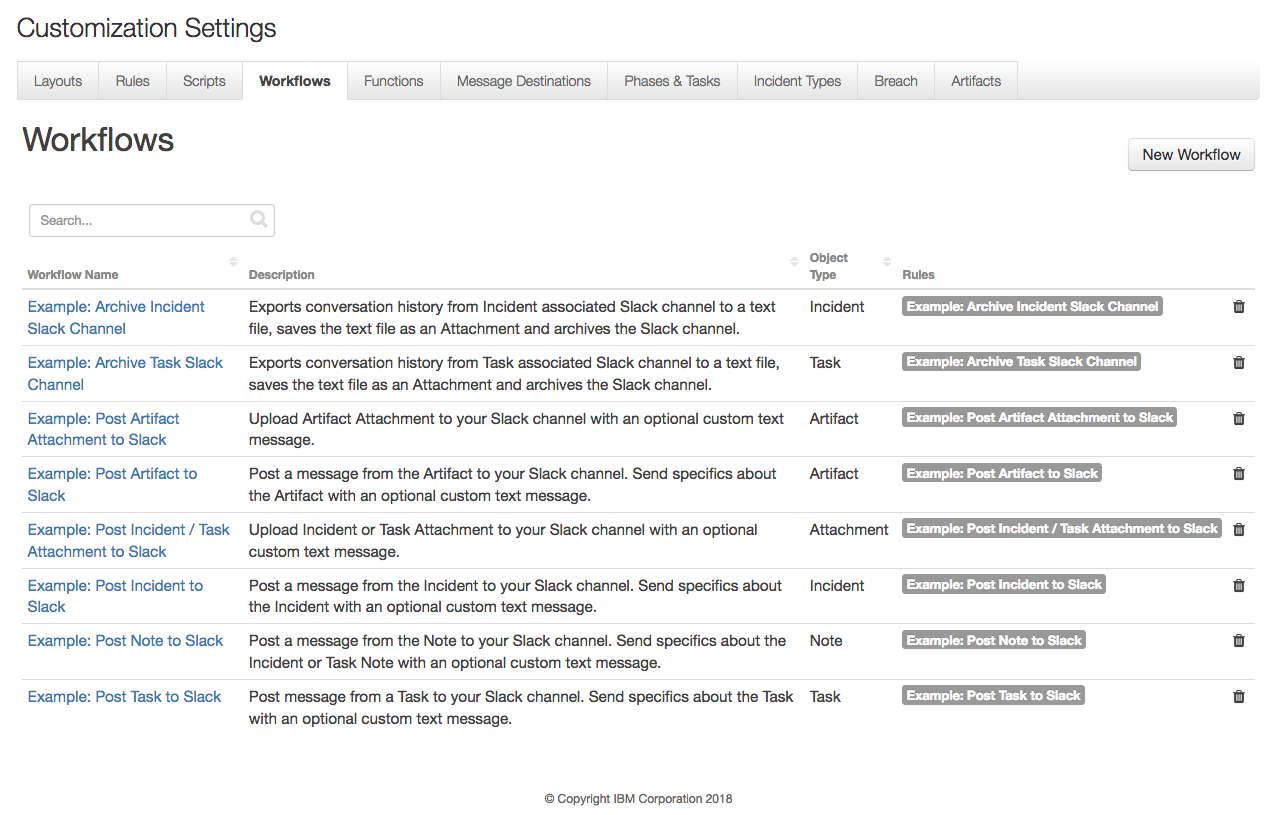
selftest: success, Elapsed time: 0.000000 seconds

Function Descriptions

Once the function package deploys, you can view the three functions in the Resilient platform Functions tab, as shown below.



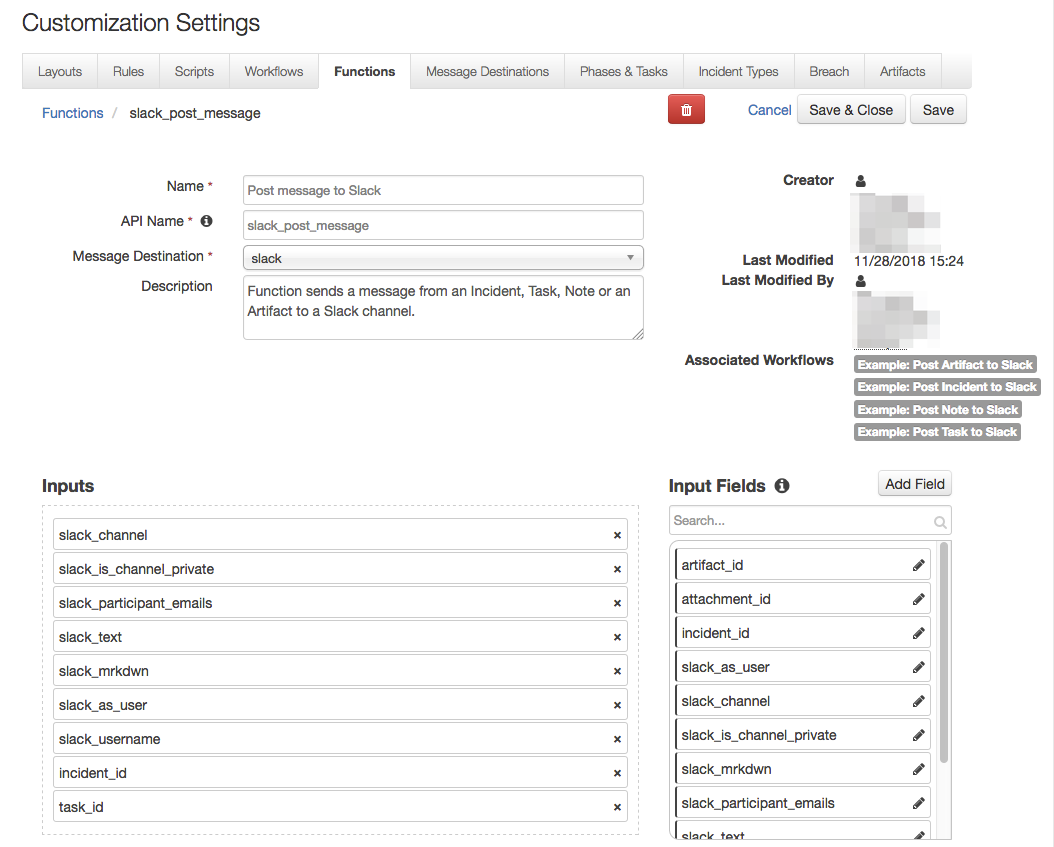
The package includes example workflows and rules that show how the functions can be used. You can copy and modify these workflows and rules for your own needs.



Function: Post message to Slack

This function posts data from an Incident, Task, Note, or an Artifact to a Slack channel, and takes the following input fields:

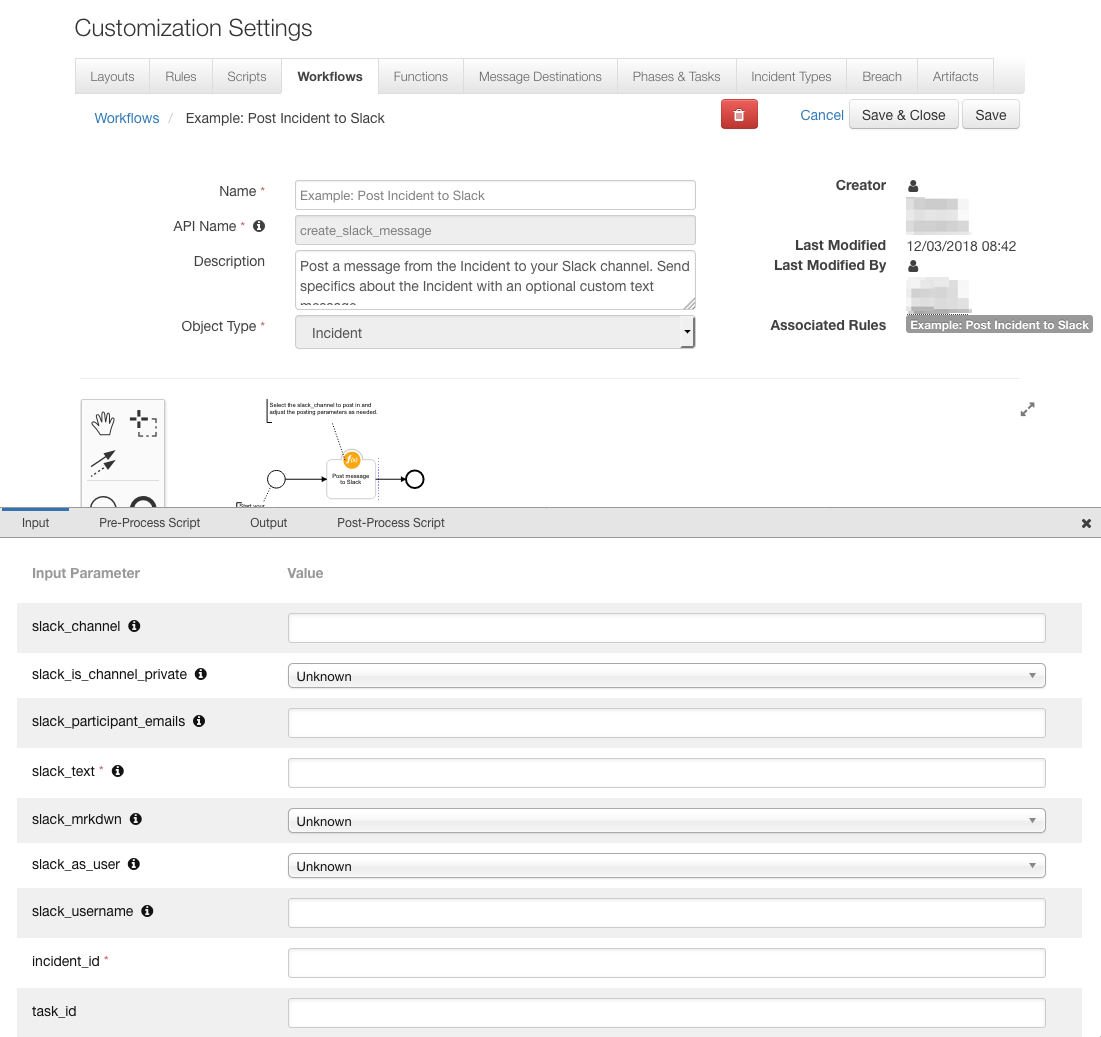
* slack\_channel: Name of the existing or a new Slack channel, where the app will post data. Channel names can contain only lowercase letters, numbers, hyphens, and underscores, and must be 21 characters or less. If you leave this field empty, the function tries to use the slack\_channel associated with the Incident or Task found in the Slack Conversations Data Table. If one is not defined, the workflow terminates.
* slack\_is\_channel\_private: Determines if the channel you are posting to should be private.
* slack\_participant\_emails: Comma-separated list of emails belonging to Slack users in your workspace that will be added to your channel.
* slack\_text: A text message or a container field to retain JSON fields to send to Slack.
* slack\_mrkdwn: Disables Slack markup parsing by setting to False.
* slack\_as\_user: If True, the authenticated user of the Slack App appears as the author of the message, ignoring any values provided in slack\_username.
* slack\_username: Replaces your Slack App’s name to appear as the author of the message. Must be used in conjunction with slack\_as\_user set to False; otherwise, it is ignored.



The following example workflows call this function to post object data to a Slack channel:

* Example: Post Incident to slack
* Example: Post Task to Slack
* Example: Post Note to Slack
* Example: Post Artifact to Slack

The workflows can set the input field values from the Input tab.

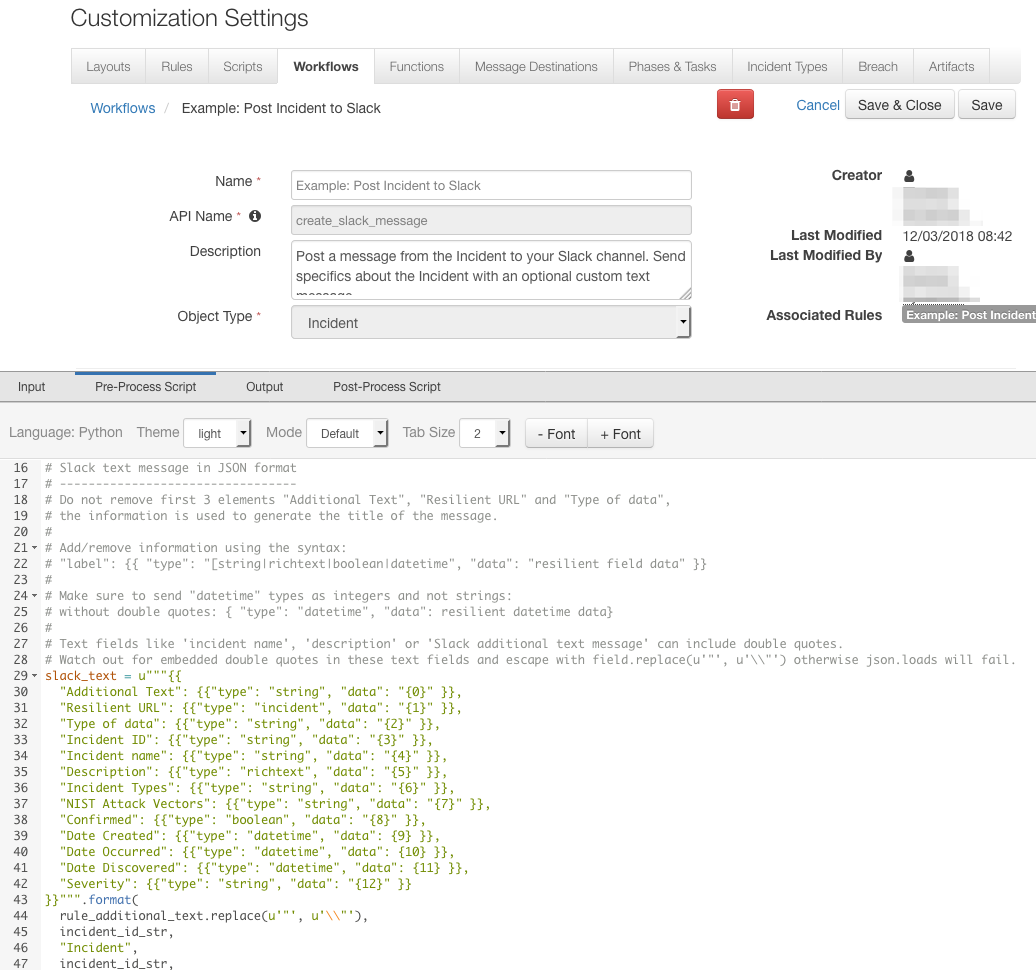


The default settings for posting messages in the function are:

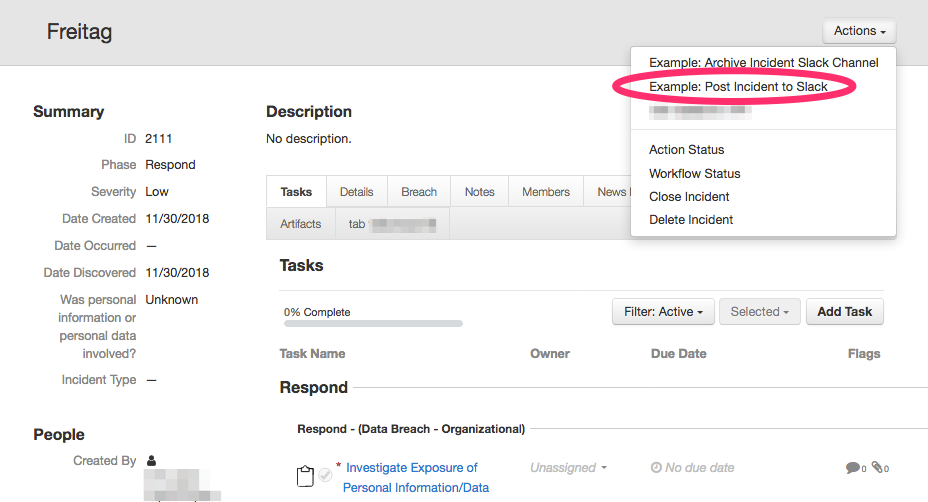
* parse="none": Slack does not perform any processing on the message. It keeps all markup formatting.
* link\_names=1: Slack converts URLs, channel names (starting with a '#') and username IDs (starting with '<@ user\_id >') to links. The function can send messages like, “Hey user <@UCNC5K34J> check out #random”, to Slack and it finds and links channel names and usernames.

Refer to the [Slack API chat.postMessage documentation](https://api.slack.com/methods/chat.postMessage) for a detailed explanation of the Slack arguments.

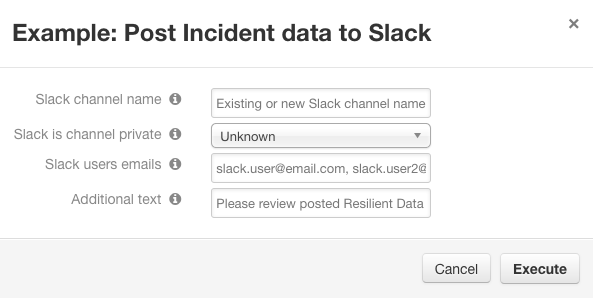
In the workflow’s Pre-Process Script tab, the data to post is customizable. A flexible JSON structure is used to define the Incident, Task, Note and Artifact fields to post in Slack. An additional text message, set in the Menu Item, can also be included with the data posted. Fields can be removed and added following the defined data structure.



Some of the function input fields can be set by clicking a Menu Item rule for the object type.

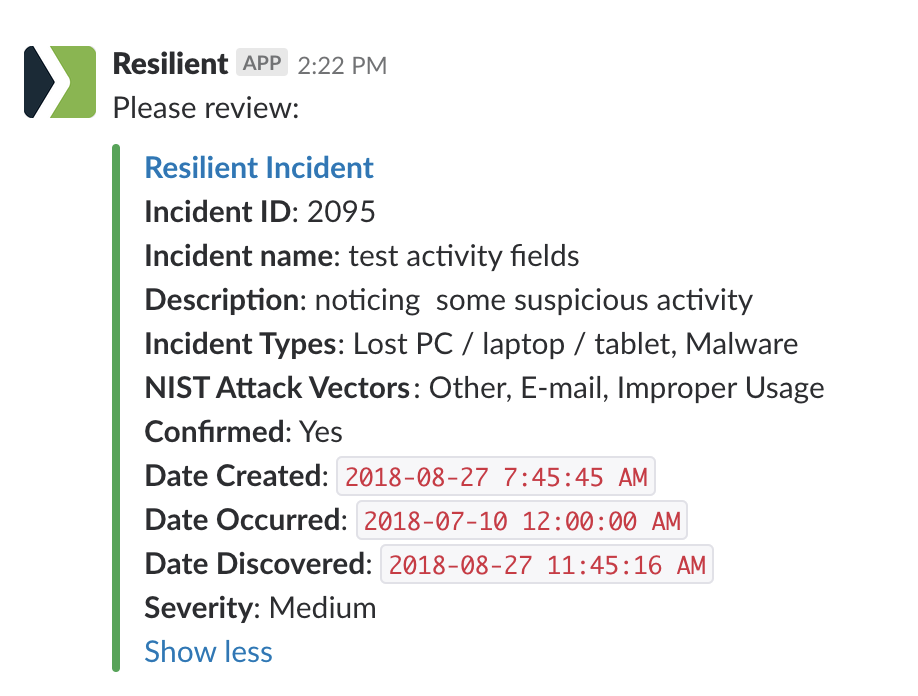


These values either override those set in the workflow or are included with the message posted in Slack:



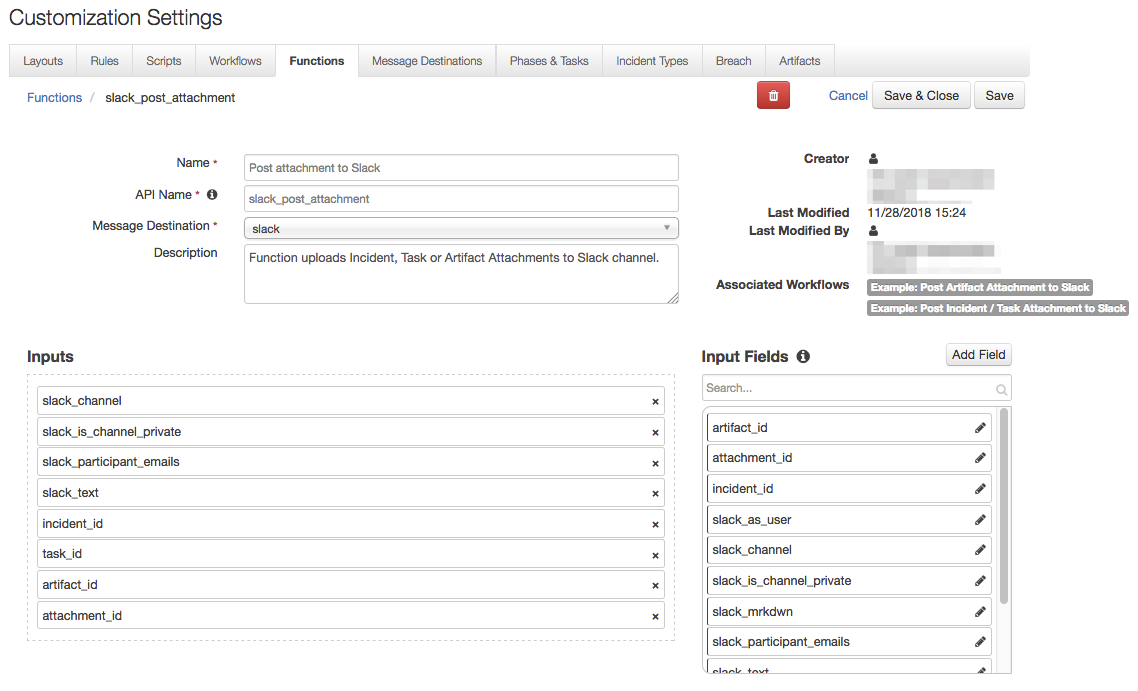
* The values in Slack channel name, Slack is channel private and Slack user emails, override those set in the workflow.
* Additional text: Includes the text in this field with the Incident, Note, Artifact, Attachment or Task data to send to Slack.

Your posted message will look like this in Slack:



Function: Post attachment to Slack

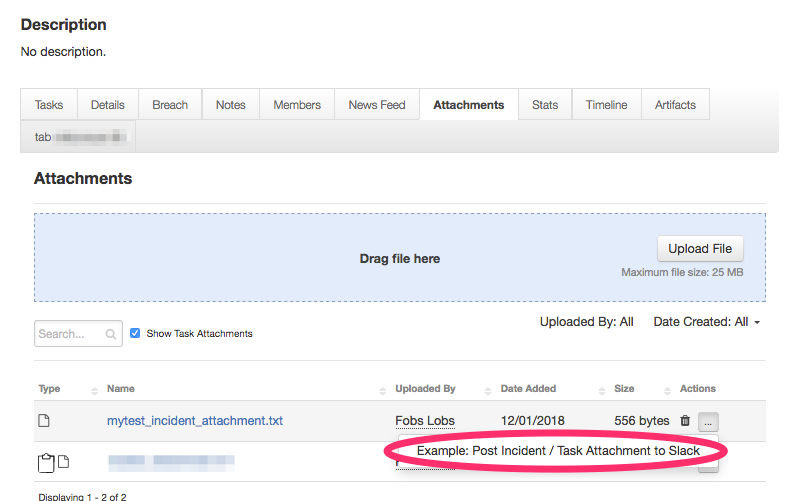
This function uploads attachments from an Incident, Task, or Artifact to a Slack channel.

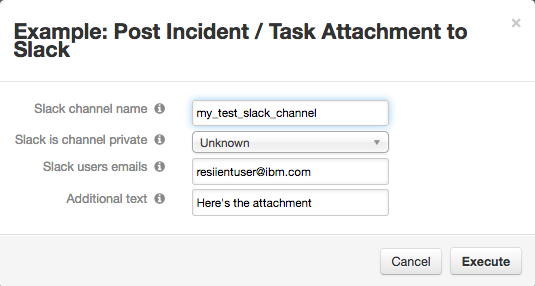


The workflows associated with this function are:

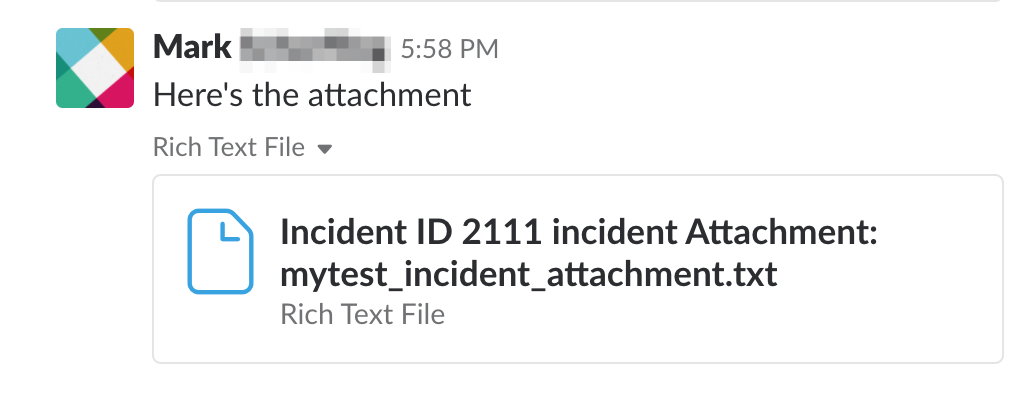
* Example: Post Incident/Task Attachment to Slack
* Example: Post Artifact Attachment to Slack

The function input fields can be set on the Input Tab of the workflow or when you click the Menu Item for the object, which is the same way as shown for the Post message to Slack Function.





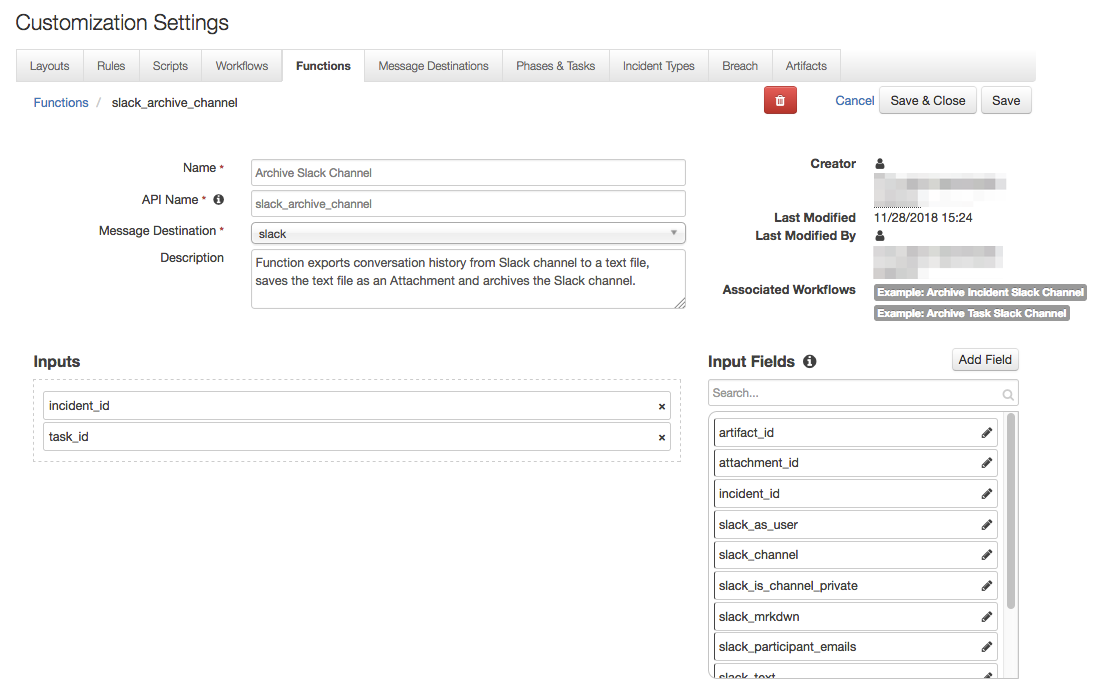
Your posted message with an upload attachment will look like this in Slack:



To upload files, the function uses the [Slack API files.upload method](https://api.slack.com/methods/files.upload), which uses the Slack App OAuth Token that belongs to the authenticated user of the Slack App. File uploads sent using this token are uploaded on behalf of the user - not the Slack App. This behavior can be changed by adding a [Bot User](https://api.slack.com/bot-users) to your Slack App and using Bot User OAuth Token for file uploads.

Function: Archive Slack Channel

This function exports a conversation history from a Slack channel to a text file, saves the text file as a Resilient attachment, and archives the Slack channel.



The following example workflows demonstrate archiving a Slack channel from an Incident or Task:

* Example: Archive Incident Slack Channel
* Example: Archive Task Slack Channel

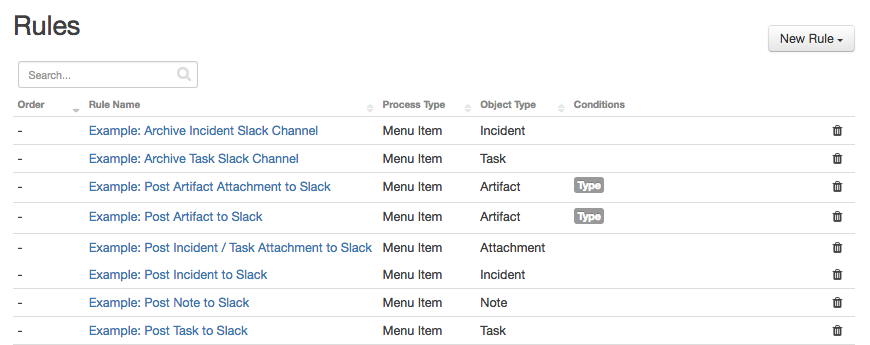
When invoked, the Archive Slack Channel function searches for the associated channel name in the Slack Conversations Data Table. If an Incident or Task has an existing connection with a slack\_channel, the function archives this channel.

The exported conversation history looks like this:

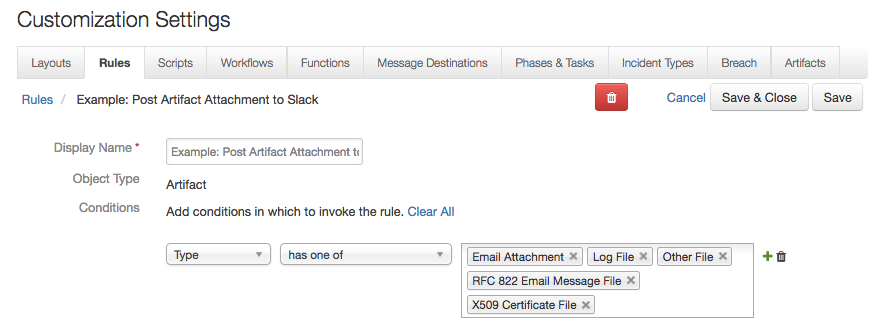


Resilient Platform Configuration

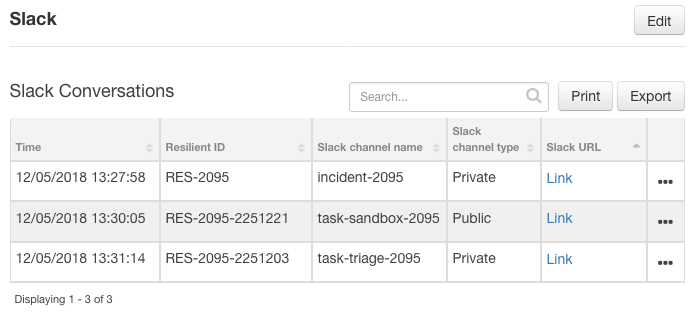
The following eight rules, which you can customize, are defined in the package:



NOTE: The rule, Example: Post Artifact Attachment to Slack, allows uploading attachments from only certain types of artifacts. Artifact type Malware sample is excluded because of the potential risk of sending malware samples to your Slack channel.



There is also a Slack Conversations Data Table, slack\_conversations\_db, created in the resilient-circuits customize step. Users may add this data table to a custom layout.



The purpose of the data table is to save connections between an Incident and a Slack channel or a Task and a Slack channel. An Incident and each Task can each have one associated channel – the default channel, but users may also post to other channels – separate non-default channels. The function only saves the connection to the first Slack channel that gets a post.

When invoking the Post message to Slack or Post attachment to Slack functions, users can specify whether they are posting to their associated - default Slack channel or to a separate - non-default channel.

If the function input field, channel\_name, is not specified either on the Menu Item or on the Input tab of the workflow, the function searches for the associated channel name in the Slack Conversations Data Table. If the Incident or Task has an existing connection with a slack\_channel, the function finds this channel in your workspace and posts there.

If channel\_name input field is specified, the function tries to find this channel in your workspace and posts there. If the channel does not yet exist, it creates a new channel.

If channel\_name input field is specified and the Incident or Task already has an existing connection with a slack\_channel, the function ignores the associated one in the data table and posts to the input one.

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 actions 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, go to <https://www.ibm.com/mysupport>.

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

Template file

If you’d like to customize the template file, use the template\_file app.config setting to specify the location of your jinja2 template. Below is the default template.

{%- if is\_msg\_parent -%}  
 {{- number }} - {{ username }} POSTED ON {{ msg\_time }}:{{ '\n' -}}  
 {%- if msg\_pretext -%}  
 {{- msg\_pretext }}{{ '\n' -}}  
 {%- endif -%}  
 {%- if msg\_text -%}  
 {{- msg\_text }}{{ '\n' -}}  
 {%- endif -%}  
 {%- if file\_name -%}  
 {{- file\_name }}{{ '\n' -}}  
 {%- endif -%}  
 {%- if file\_permalink -%}  
 {{- file\_permalink }}{{ '\n' -}}  
 {%- endif %}{{ '\n' -}}  
 {%- if reply\_count and reply\_count > 0 -%}  
 {{- reply\_count }} REPLIES:{{ '\n' -}}  
 {% endif -%}  
{%- else -%}  
 {{- '\t' }}{{ number }} - {{ username }} POSTED A REPLY ON {{ msg\_time }}:{{ '\n' -}}  
 {%- if msg\_text -%}  
 {{- '\t' }}{{ msg\_text }}{{ '\n' -}}  
 {%- endif -%}  
 {%- if file\_name -%}  
 {{- '\t' }}{{ file\_name }}{{ '\n' -}}  
 {%- endif -%}  
 {%- if file\_permalink -%}  
 {{- '\t' }}{{ file\_permalink }}{{ '\n' -}}  
 {% endif %}{{ '\n' -}}  
{%- endif -%}