

Developers Hub

Version 1 — Last update: Apr 20, 2022

Siemplify

Table of Contents

1. Start Developing in Siemplify	4
1.1. Siemplify Integration Marketplace.....	5
1.2. Getting Started with Siemplify	8
1.3. My First Integration	13
1.4. My First Action	22
1.5. My First Connector.....	34
1.5.1. Developing the Connector.....	37
1.5.2. Configuring the Connector	52
1.5.3. Testing the Connector.....	54
1.5.4. Mapping & Modeling	58
1.6. My First Automation	63
1.7. Publish Your First Integration	76
1.8. Requirements for Publishing Integration.....	78
1.9. My First Use Case.....	89
1.9.1. Creating a Use Case.....	90
1.9.2. Requirements for Publishing Use Case	100
2. Playbook Lifecycle Management	101
2.1. Prerequisites	102
2.2. Basic Playbook Design.....	103
2.2.1. Know your Alerts	104
2.2.2. Analyze existing manual flow	106
2.2.3. Begin Playbook Design	107
2.2.4. Playbook Blocks – Identify Repeatable Logical Flows	109
2.2.5. Playbook Block Design	110
2.2.6. Design Tips.....	111
2.2.7. Summary of Implementation.....	112
2.3. Build the Playbook Block.....	113
2.3.1. Determine Playbook block output.....	114
2.4. Build Playbook	115
2.5. Individual Features.....	116
2.5.1. Placeholders and the Expression Builder	117
2.5.2. Entities.....	122
2.5.3. Conditions.....	123
2.5.4. Error Handling.....	124
2.5.5. Environments.....	125
2.5.6. Insights	126
2.5.7. Simulate Alerts.....	127
2.5.8. How Playbooks work behind the scenes	128
3. Siemplify API	129

4. SDK References	130
4.1. Concepts & Tutorials	131
4.1.1. Actions	132
4.1.1.1. Action Results	133
4.1.2. Integration Configuration & Script Parameters	137
4.1.2.1. External Configuration Providers	138
4.1.3. Custom Lists	142
4.1.4. Case Manipulation	144
4.1.4.1. Insights (General/Entity)	146
4.2. API	148
4.2.1. SiemplifyBase (SiemplifyBase.py)	149
4.2.1.1. fetch_timestamp	150
4.2.1.2. save_timestamp	151
4.2.1.3. fetch_and_save_timestamp	152
4.2.1.4. run_folder	153
4.2.2. Siemplify (Siemplify.py)	154
4.2.2.1. add_Attachment	155
4.2.2.2. add_comment	156
4.2.2.3. add_entity_insight	158
4.2.2.4. add_entity_to_case	160
4.2.2.5. add_entities_to_custom_list	162
4.2.2.5.1. extract_configuration_param	164
4.2.2.6. any_entity_in_custom_list	166
4.2.2.7. assign_case	168
4.2.2.8. attach_workflow_to_case	170
4.2.2.9. change_case_priority	171
4.2.2.10. create_case	173
4.2.2.11. end	174
4.2.2.12. end_script	176
4.2.2.13. get_case_comments	177
4.2.2.14. get_existing_custom_list_categories	179
4.2.2.15. is_existing_category	181
4.2.2.16. mark_case_as_important	183
4.2.2.17. raise_incident	184
4.2.2.18. remove_entities_from_custom_list	185
4.2.2.19. update_entities	187
4.2.3. SiemplifyAction (SiemplifyAction.py)	188
4.2.3.1. add_attachment	189
4.2.3.2. add_comment	191
4.2.3.3. add_entity_to_case	192
4.2.3.4. add_alert_entities_to_custom_list	195
4.2.3.5. add_tag	197


4.2.3.6. any_alert_entities_in_custom_list	199
4.2.3.7. assign_case.....	201
4.2.3.8. attach_workflow_to_case.....	203
4.2.3.9. change_case_priority.....	204
4.2.3.10. change_case_stage.....	206
4.2.3.11. close_case.....	208
4.2.3.12. close_alert.....	210
4.2.3.13. create_case_insight.....	212
4.2.3.14. extract_action_param	214
4.2.3.15. get_alerts_ticket_ids_from_cases_closed_since_timestamp	216
4.2.3.16. get_attachments	217
4.2.3.17. get_case_comments.....	218
4.2.3.18. get_configuration	220
4.2.3.19. get_similar_cases	222
4.2.3.20. load_case_data	224
4.2.3.21. mark_case_as_important.....	225
4.2.3.22. raise_incident	226
4.2.3.23. remove_alert_entities_from_custom_list	227
4.2.3.24. set_logs_collector.....	229
4.2.3.25. update_alerts_additional_data	230
4.2.3.26. _get_case.....	231
4.2.3.27. _load_current_alert.....	232
4.2.3.28. _load_target_entities	233
4.2.3.29. _get_custom_list_items	234
4.2.4. SiemplifyConnectorExecution (SiemplifyConnectors.py)	235
4.2.4.1. is_overflowed_alert.....	236
4.2.4.2. return_package.....	238
4.2.4.3. return_test_result.....	239
4.2.4.4. extract_connector_param	240
4.2.5. SiemplifyJob (SiemplifyJob.py)	242
4.2.5.1. get_configuration	243
4.2.5.2. extract_job_param	245
4.2.5.3. get_system_info	247
4.2.6. ScriptResult (ScriptResult.py)	249
4.2.6.1. add_entity_json	250
4.2.6.2. add_result_json	251
4.2.6.3. add_entity_content	252
4.2.6.4. add_entity_table	253
4.2.6.5. add_entity_attachment.....	254
4.2.6.6. add_entity_html_report	256
4.2.6.7. add_entity_link	257
4.2.6.8. add_link.....	258

4.2.6.9. add_attachment.....	259
4.2.6.10. add_content.....	260
4.2.6.11. add_html.....	261
4.2.6.12. add_json.....	262
4.2.6.13. add_data_table.....	264
4.2.7. SiemplifyLogger (SiemplifyLogger.py)	265
4.2.7.1. loadConfigFromFile	266
4.2.7.2. exception.....	267
4.2.7.3. error	269
4.2.7.4. warn	270
4.2.7.5. info	271

1. Start Developing in Siemplify

1.1. Siemplify Integration Marketplace

The Siemplify Marketplace allows you to find and install an integration of third party applications, custom integrations that you have built in the IDE, and pre-built playbook workflows to integrate into the organizational security products for automated IR process and optimize your Siemplify installation. The Marketplace also contains a repository for use cases – including predefined use cases from Siemplify and customer uploaded use cases.

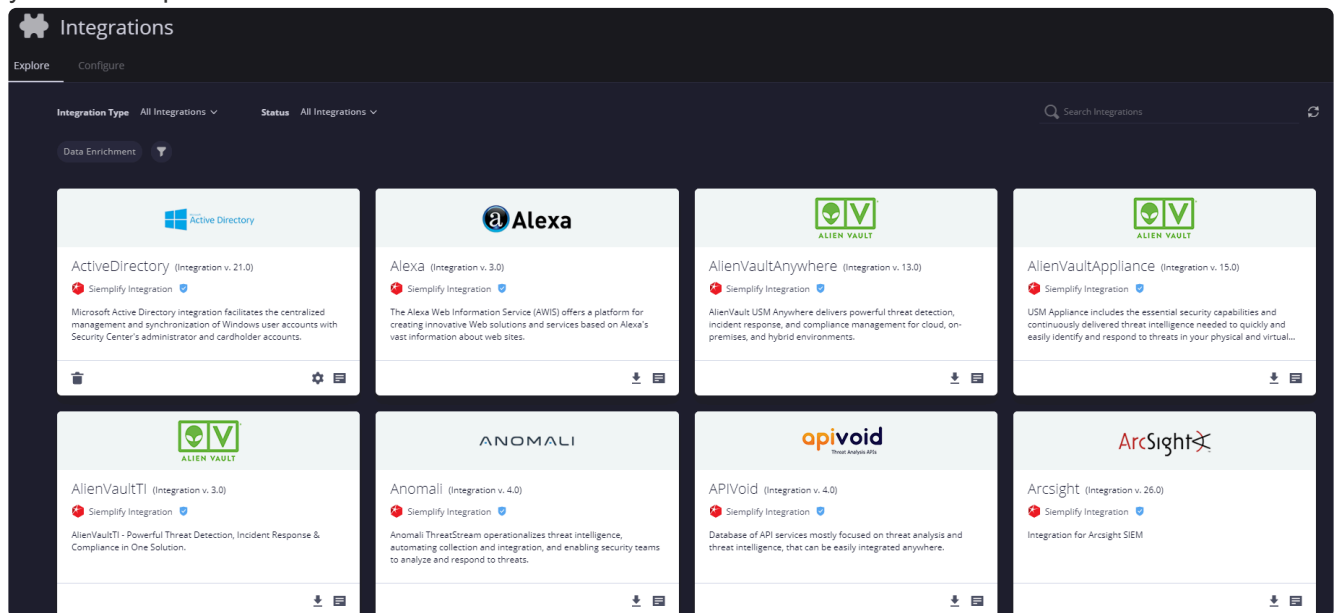
Clicking on the Marketplace icon  on the top right of the screen allows you to choose between Integrations and Use Cases.

Integrations

Clicking on Integrations displays the following screen.

There are three types of integrations you can see in the Marketplace:

- Siemplify Integrations
- Integrations published by users (which have been validated by Siemplify and which will appear with user details next to them)
- Custom Integrations (these are integrations which you have created and which are only displayed on your Marketplace).



Integrations Explore

You can display the Integrations according to integration type (for example, show custom integrations, published by users) or by status (for example, installed, available update).

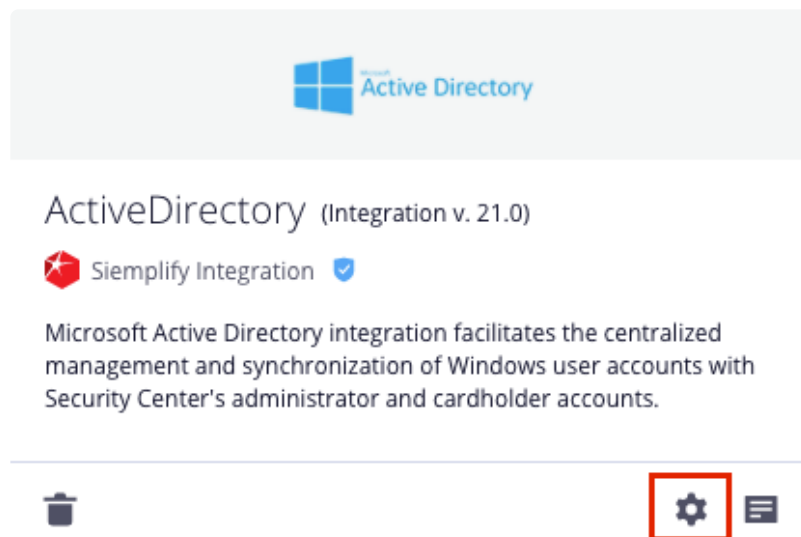
Integrations that have not been installed yet will have a downwards arrow on the bottom right of the box.

Click on this to successfully install the integration. Custom Integrations will not show the downwards arrow as they are installed via the IDE. All integrations need to be configured and saved. For detailed information on installing and configuring an Integration, see [here](#).

Note that for each supported Integration in the Marketplace, there will be a link to an [Integrations and Connectors Portal](#) page with detailed information on that specific Integration.

Integrations Configure

1. There is an option to configure each integration under a default environment by clicking on the gear icon after you downloaded the integration.



The screenshot shows a card for the 'ActiveDirectory' integration. At the top is the Microsoft Active Directory logo. Below it, the text 'ActiveDirectory (Integration v. 21.0)' is displayed. Underneath is a red star icon followed by 'Simplify Integration' and a blue checkmark. A descriptive paragraph follows: 'Microsoft Active Directory integration facilitates the centralized management and synchronization of Windows user accounts with Security Center's administrator and cardholder accounts.' At the bottom, there is a horizontal bar with three icons: a trash can, a gear icon (highlighted with a red square), and a menu icon.

2. Clicking on the gear icon will open the configuration window and will present all the fields related to the integration that are required to configure for successful connection to the product.

ActiveDirectory - Configure Instance

Configure all the necessary fields and parameters for this instance

Environment **DE Default Environment**

Instance Name System Default Instance

Description

Parameters

For more information on configuration and integration details, [click here](#)

Server * x.x.x.x

Username * user@domain.com

Domain * domain.com

Password *

Custom Fields customField1,customField2

Use SSL ☐

Remote Agent

Run Remotely ☐

No agents configured, [Install agent](#).

Test Cancel Save

- If you would like to configure an integration under a different instance, click on the configure tab and choose the instance you would like to configure the integration to. For more information on Simplify Instances, you can access the [“Supporting Multiple Instance”](#) guide.

Integrations

Explore **Configure**

Environments (2)

☐ Hide empty environments

Search...

Shared Instances
2 integrations

DE Default Environment
8 integrations

Shared Instances
Add and configure instances that will be shared across all environments

Search...

Simplify		1 instances
Simplify_1	This instance was created by Simplify	Configured
SimplifyUtilities		1 instances
SimplifyUtilities_1	This instance was created by Simplify	Configured

1.2. Getting Started with Siemplify

To start SOARing in the Siemplify platform, the first thing to understand is the very basic concepts which are mentioned frequently in our documentation, and are important to know.

Be sure to read it in the given order, since each explanation relies on the prior one.

Connectors

Connectors are the ingestion point for alerts into Siemplify. Their goal is to translate raw input data, coming from various sources, into Siemplify data. The connector gets alerts (or equivalent data) from 3rd party tools, and forwards normalized data into the Data Processing layer. Siemplify provides out-of-the-box connectors for today's most popular security systems and also a Python SDK to develop new ones easily, if needed.

Cases, Alerts and Events

Case consists of alerts which were ingested from a variety of sources by the connectors. Each Alert contains one or more security events. Once ingested into Siemplify these events are then analyzed and their indicators, destinations, artifacts, etc are extracted into objects in Siemplify called entities.

The screenshot displays the Siemplify user interface. On the left, a sidebar lists 9 cases, with 'SUSPICIOUS PHISHING EMAIL1' highlighted. The main panel shows the details of this case, including its ID (31736), location (Singapore BU), and creation time (2021-08-25 20:27:17). Below this, a list of alerts is shown, with the first alert, '1. SUSPICIOUS PHISHING ...', selected. The 'Overview' tab is active, displaying a 'Pending Actions' section with a task to 'Share IOC details with user?'. Below this, 'Entities Highlights' are listed, including a hash, a malicious count, an email address, a salary notification, and a URL. On the right, a 'Mitre Att&ck' section provides information about 'Phishing', including its description and detection methods.

SUSPICIOUS PHISHING EMAIL1

ID 31736 Singapore BU 2021-08-25 20:27:17 Triage

Data Exfiltration X ThreatFuse X +3 more Manage Tags

John Perkins

Explore

1. SUSPICIOUS PHISHING ... (1) 2021-08-10 16:49:31

2. SUSPICIOUS PHISHING ... (1) 2021-08-10 16:49:31

3. SUSPICIOUS PHISHING ... (1) 2021-08-10 16:49:31

4. DATA EXFILTRATION ... 2021-08-10 17:33:14 4 Alerts

Overview Events (1) Playbooks (1)

Name	Type	Source / Product	Artifacts	Port	Outcome	Time
Your New Salary ...	Phishing email det...	Phishing email det...	T1566,MONTHLYS...		allowed	2021-08-10 16:49:...

Event

Write a comment...

Entities

Entities are objects that represent points of interest extracted from alerts (IOCs, artifacts etc.).

Entities allow you to automatically track their history, group alerts without human intervention and hunt for malicious activity based on the relationship between the different entities.

In order to visually present the entities and their connection in the platform, there is a configuration process of the ontology that involves mapping and modelling. During this process, you select the visual representation of alerts and the Entities that should be extracted from it.

Siemplify provides basic Ontology rules for most popular SIEM products out-of-the-box.

Entities Highlights (7)

4F700AA822F77DC376C2F9D80DF9C65F2A0868D358C8EF33420B63DACBCD827F View more

vt3_malicious_count 33

F.ATTACKER4@GMAIL.COM View more

YOUR NEW SALARY NOTIFICATION View more

HTTP://MARKOSSOLOMON.COM/F1Q7QX.PHP View more

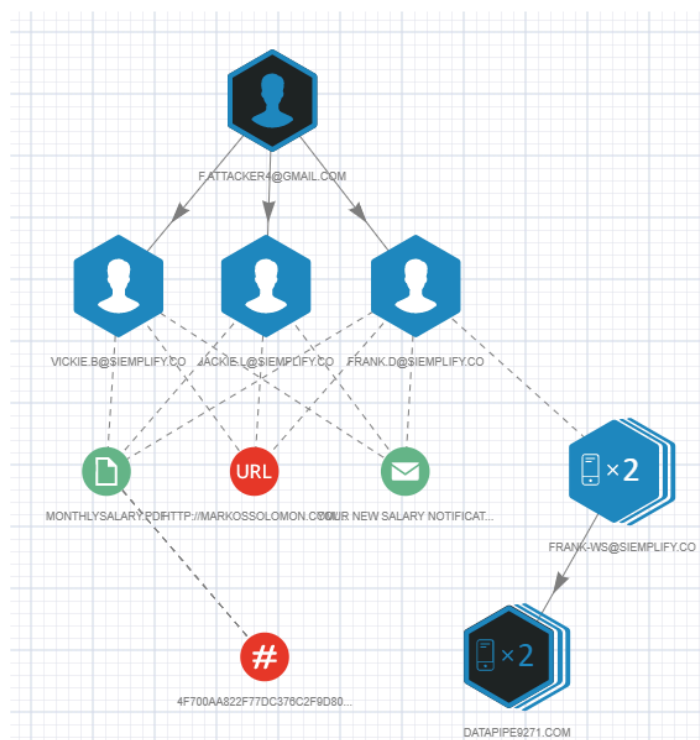
T1566 View more

How to create Entities in Siemplify

The process of mapping & modeling allows you to create the entities related to a specific model family and to visualize the connections between them in a case.

The screenshot shows the Siemplify Event configuration interface. The top navigation bar includes links for HOMEPAGE, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, REPORTS, and COMMAND CENTER. The main header indicates the current configuration is for the 'Phishing email detector' under the 'Arcsight' product. On the left, there are tabs for 'Visualization' and 'Mapping'. The 'Model families' section is active, showing a list of model families. The 'MailRelayOrTAP' family is selected, showing a visual representation of an email monitoring event. Below this, there are several other model families, each with a 'Custom' button: File Extraction (DLP), DLP Files (Files being exfiltrated), IOC (IOC), Custom MailRelayOrTAP (Email monitoring event), ActiveDirectory user creation (Suspicious Active Directory activity), DLP (Unauthorized data exfiltration), DAM (Suspicious user activity on a database object), and ActiveDirectory (Suspicious Active Directory activity).

By mapping & modelling we can define the entity properties such as what defines if an entity is internal or external is the configuration in the settings and if its malicious or not by the product we run in the playbook. The mapping and modeling is more for what is source, what is time, types etc.



The mapping and modeling occurs once during the first time it is ingested and from then on the rules will run on each case inserted that is relevant to it.

Playbooks

A Playbook is an automation process that can be triggered by a predefined trigger. For example, you can trigger a playbook for each alert that contains the product name “Mail”. This means that the playbook will attach to each Alert ingested into Siemplify from this product.

The screenshot shows the Siemplify Playbook configuration interface. The top navigation bar includes Home, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, REPORTS, and COMMAND CENTER. The left sidebar shows the 'Step Selection' panel with various triggers and actions. The main area displays the 'Phishing' playbook configuration. The 'Triggers' tab is selected, showing a list of triggers including 'Alert Trigger Value', 'Alert Type', 'All', 'Custom List', 'Custom Trigger', 'Network Name', 'Product Name', and 'Tag Name'. The 'Alert Type' trigger is highlighted with a red box. The 'Actions' tab shows a sequence of actions: 'Alert Type', 'Siemplify Change Case...', 'AD Enrichment Simula...', 'Mitre 1', and 'Automated Trigs...'. The 'Alert Type' action is highlighted with a red box. The 'Run' button is visible at the bottom right.

Each playbook consists of actions that can be configured to run manually or automatically on the scope defined for the alert entities. For example, we can configure VirusTotal – Scan URL action to run automatically only on a specific entity type such as URL entities.

The screenshot shows the 'VirusTotal - Scan URL' configuration dialog. The 'Action Type' is set to 'Automatic'. The 'If step fails' option is 'Stop playbook'. The 'Choose Instance *' is 'Default Environment_System Default Instance'. The 'Entities' field is set to 'All URLs' and is highlighted with a red box. The 'Rescan after days' is set to '100' and the 'Threshold *' is set to '2'. The 'Save' button is visible at the bottom right.

The actions take place by their defined order according to conditions- (flows) forming a tree of actions.

When the final action is done, the playbook gets to a resolution for the triggering alert.

1.3. My First Integration

Overview

Siemplify users can create custom integrations inside the Siemplify IDE with the same structure as Siemplify commercial integrations. The custom integrations will appear in the Marketplace and can be configured for different environments so they can be used in Playbooks, manual actions and remote agents. They can also be imported and exported as with other IDE items.

In this How To we will build a custom integration for the “WHOIS XML API” product. We will start off by creating your first integration including the registration process to the WHOIS product and the creation of the API key.



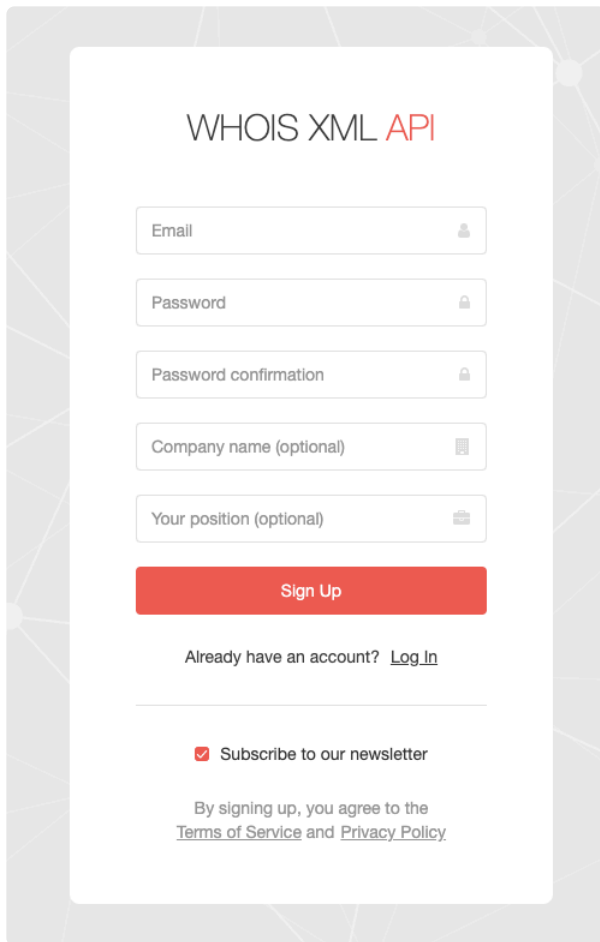
<https://www.youtube.com/embed/1u80zK-edpw?rel=0>

Choose the product you would like to integrate with

1. For this example we have chosen to integrate with “WHOIS XML API” product, a free open source tool which gets API access to domain data, including the registrant name, organization, e-mail address, registration address, registrar information, creation date, expiration date, updated date, domain

availability, domain age and many more.

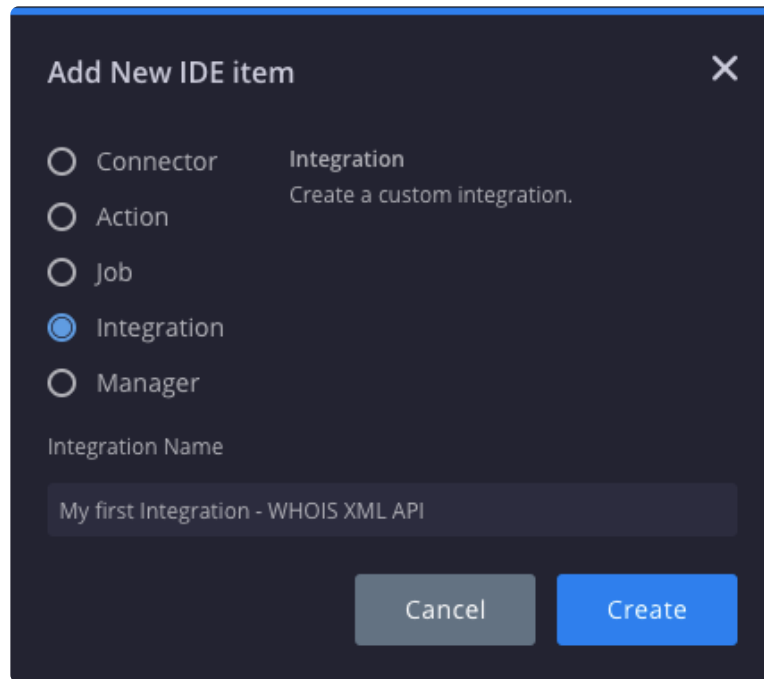
2. Lets start off by registering to WHOIS product by accessing the following url – <https://www.whoisxmlapi.com/>

The image shows a registration form for WHOIS XML API. The form is titled "WHOIS XML API" with "API" in red. It contains several input fields: "Email" with an envelope icon, "Password" with a lock icon, "Password confirmation" with a lock icon, "Company name (optional)" with a document icon, and "Your position (optional)" with a person icon. Below these fields is a red "Sign Up" button. Under the button, there is a link "Already have an account? Log In". At the bottom, there is a checkbox labeled "Subscribe to our newsletter" which is checked. Below the checkbox, there is a line of text: "By signing up, you agree to the Terms of Service and Privacy Policy".

3. After you login you can extract your API key from the following url – <https://user.whoisxmlapi.com/products>
4. Now that you have your API Key we will use this key in the integration parameters in your first custom integration.

Creating your first custom integration in the IDE

1. From the IDE screen click the + icon in the upper left hand corner to add a new IDE item. Select the Integration radio button and give the integration a name.



A dark-themed dialog box titled "Add New IDE item" with a close button (X) in the top right corner. It contains five radio button options: "Connector", "Action", "Job", "Integration" (which is selected with a blue dot), and "Manager". To the right of the "Integration" option is the text "Integration" and "Create a custom integration." Below the options is a text input field labeled "Integration Name" containing the text "My first Integration - WHOIS XML API". At the bottom right are two buttons: "Cancel" and "Create".

Add New IDE item ✕

☐ Connector

☐ Action

☐ Job

☒ Integration

☐ Manager

Integration

Create a custom integration.

Integration Name

My first Integration - WHOIS XML API

Cancel Create

2. The integration will be created and listed on the left hand side with an infinity icon that designates it as a custom integration. Clicking on the Gear Icon will bring up the Integration Settings where the Icon, Description, Python Dependencies and Integration Parameters can be defined.


My first Integration - WHOIS XML API

Running on Python V3.7


Description

Whois API provides consistent, well-structured whois data in XML & JSON. It provides WHOIS record and domain related information.

Svg icon



Image



Categories

Data Enrichment

Libraries

Add

Script Dependencies

No dependencies yet

Parameters

Parameter	Type	Default Value	Mandatory	Description
API Key	Password	*****	✓	

Close

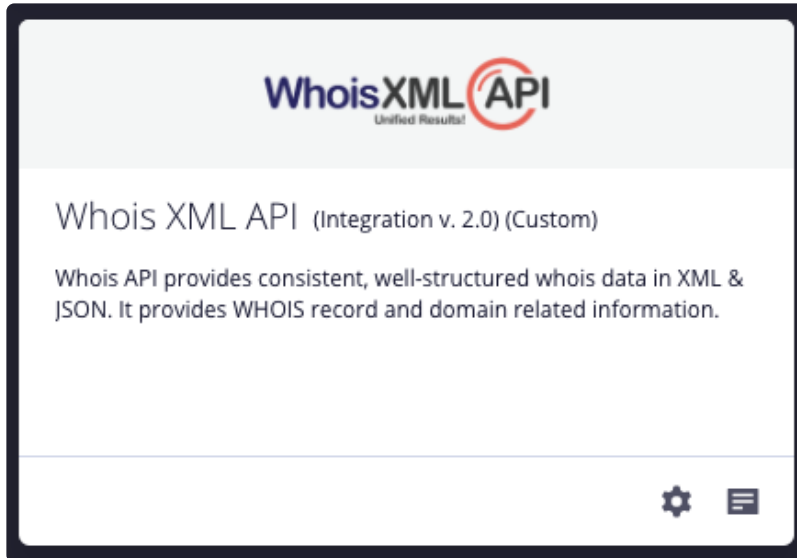
Save

3. In the following screenshot, an [image](#) has been uploaded (this image will appear in the Marketplace with the integration), an [SVG icon](#) has been added and will be presented next to the integration in the IDE, a brief description has been added and one parameter. The parameter added is the API Key which the “Who Is XML API” Product requires for the configuration of the integration. There is no need for additional python libraries for this integration. In addition, you will see that we chose to run the integration on Python 3.7. You can customize this by clicking on the dropdown and selecting to run an integration on Python 2.7.

✿ Script Dependencies are Python libraries that the custom integration will need to import. Dependencies can be added as wheel files, tarballs, gunzip format or python files (.whl, .tar, .gz, .py extensions are supported). Every integration runs in its own virtual environment so feel free to add different versions of libraries even if one is already installed on the system. For example, if there is a newer (or older!) version of requests that you would like to use

instead of the default on the system (2.20.0 at the time of writing), download the dependency from a reliable source such as PyPi or GitHub and add it to the Script Dependencies for this integration. If a dependency is not installed in the virtual environment, the integration will import it from the system installation if the dependency is installed there.

4. Once you create the integration you can view it in your Siemplify Marketplace (you can search the integration name in the search bar or filter the Integration type by “Custom Integrations”) with the image, description and parameter you defined for the integration.



5. Next, select the gear icon to open up the Configure a default Instance screen. Fill in the API Key copied from the [product page](#) in the Who Is XML API website and click on save. If you would like to configure the integration to a different instance (not the default environment) click on the configure tab and configure the integration under the relevant instance.

Whois XML API - Configure Instance

×

Configure all the necessary fields and parameters for this instance

Environment

DE Default Environment

Instance Name

System Default Instance

Description

Parameters

API Key

i |

Remote Agent

Run Remotely

☐

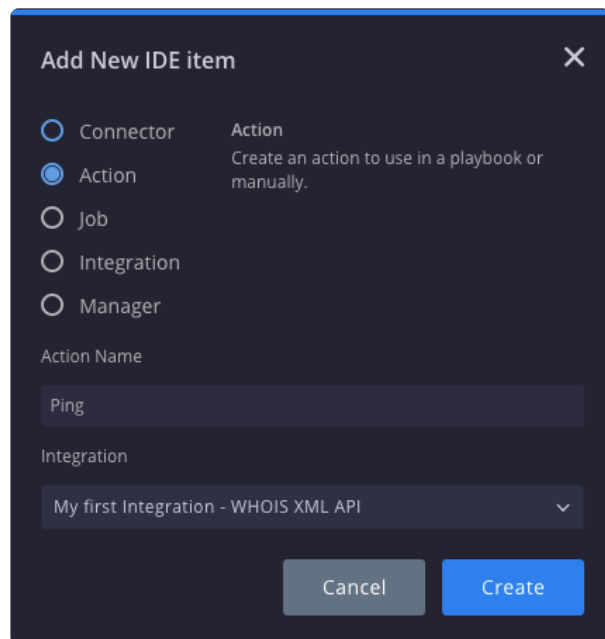
No agents configured, [Install agent](#).

Test

Cancel

Save

6. If you click on the test button in the configuration tab the test will fail. In order to make sure that you have successful authentication to the WHOIS product before you move forward to creating your first action, we will create a ping action and test the connection to the product.
7. Navigate to the IDE and click the + sign in the upper left hand corner to Add New IDE Item. Select the Action radio button, name the Action and select the integration then click the Create button.



Add New IDE item [X]

☐ Connector **Action**
Create an action to use in a playbook or manually.

☒ Action

☐ Job

☐ Integration

☐ Manager

Action Name

Ping

Integration

My first Integration - WHOIS XML API

Cancel Create

The IDE will create a new template that has some very useful code comments and explanations. Make sure to give this template a look over when possible.

8. Copy the following code for the ping action. The ping action uses the API Key parameter we configured for the integration and places that API Key in the Url provided by the product for testing purposes. We will elaborate on this in the [“My First Action”](#) tutorial.

```
from SiemplyAction import SiemplyAction
from SiemplyUtils import output_handler
import requests

INTEGRATION_NAME = "My first Integration - Whois XML API"

SCRIPT_NAME = "Whois XML API Ping"

@output_handler
def main():
    siemply = SiemplyAction()
    siemply.script_name = SCRIPT_NAME

    api_key = siemply.extract_configuration_param(provider_name=INTEGRATION_NAME,
                                                param_name="API Key")

    url = "https://www.whoisxmlapi.com/whoisserver/WhoisService?apiKey={api_key}&domainName=google.com".format(api_key=api_key)
```

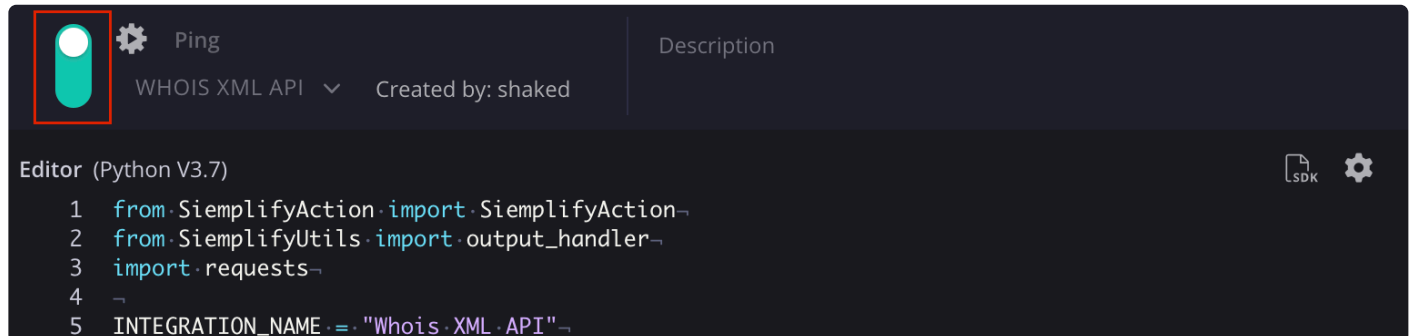
```
res = requests.get(url)
res.raise_for_status()

if "ApiKey authenticate failed" in res.content.decode("utf-8"):
    raise Exception("Error, bad credentials")

simplify.end("Successful Connection", True)

if __name__ == "__main__":
    main()
```

9. In order to test the connection to the product enable the toggle above the action and click on Save.



The screenshot shows the Simplify interface for configuring an integration. A toggle switch for the 'Ping' action is turned on (green) and is highlighted with a red box. Below the toggle, the text 'WHOIS XML API' and 'Created by: shaked' are visible. To the right, there is a 'Description' tab. Below the toggle, there is an 'Editor (Python V3.7)' section with a code editor showing Python code for the integration.

```
1 from SimplifyAction import SimplifyAction
2 from SimplifyUtils import output_handler
3 import requests
4
5 INTEGRATION_NAME = "Whois XML API"
```

10. Navigate to the Marketplace, click on the gear icon and make sure that the integration is configured and saved. Test the integration by clicking on the test button. If the connection is successful a green V will be presented next to the test. If the connection is not successful an X will be presented next to the test with the associated error.

Whois XML API - Configure Instance

Configure all the necessary fields and parameters for this instance

Environment **DE** Default Environment

Instance Name System Default Instance

Description

Parameters

API Key *i*

Remote Agent

Run Remotely ☐

No agents configured, [Install agent](#).

Test ☒ Cancel Save

Once you have finished the authentication step you can now create your first custom action in your custom integration.

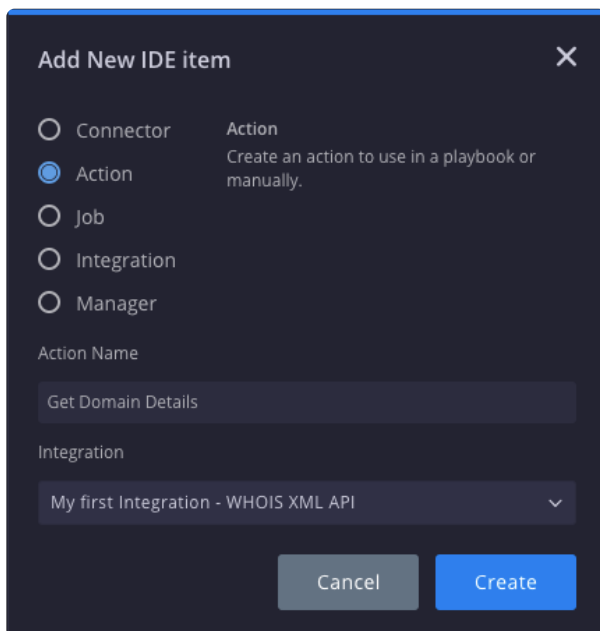
1.4. My First Action

Overview

In [My First Integration](#), we created a custom integration, defined the parameters related to the integration and created a Ping Action to test connection to the product. In this tutorial we will create two Actions for the integration, one will get the Domain details and present a Json result and the other is Enrich Entities action. Knowledge of Python and object oriented programming is necessary for this tutorial. Additionally, exploring the SDK modules themselves is highly recommended.

Creating a Custom Action

- Navigate to the IDE and click the + sign in the upper left hand corner to Add New IDE Item. Select the Action radio button, name the Action “Get Domain Details” and select the integration then click the Create button.



The screenshot shows a dark-themed dialog box titled "Add New IDE item" with a close button (X) in the top right corner. Inside the dialog, there are five radio buttons: "Connector", "Action" (which is selected and highlighted with a blue dot), "Job", "Integration", and "Manager". To the right of the "Action" radio button, there is a description: "Create an action to use in a playbook or manually." Below the radio buttons, there is a text input field labeled "Action Name" containing the text "Get Domain Details". Below that is a dropdown menu labeled "Integration" showing the selected option "My first Integration - WHOIS XML API" with a downward arrow. At the bottom of the dialog, there are two buttons: "Cancel" (grey) and "Create" (blue).

The IDE will create a new template that has some very useful code comments and explanations. Make sure to give this template a look over when possible.

Action Parameters

In order to configure the relevant parameters for this action, review the input parameters in the [WHOIS XML API documentation](#). For the Get Domain Details we will need to configure 2 parameters for the action – Check Availability & Domain Name.

da

1 results in a quick check on domain availability, **2** is slower but more accurate. Results are returned under **WhoisRecord** → **domainAvailability** (AVAILABLE | UNAVAILABLE | UNDETERMINED)

Acceptable values: **0** | **1** | **2**

Default: **0**

1. To configure the parameters click on the right + icon in the right part of the IDE module.

Details Testing Debug Output




Output Name ScriptResult

Include JSON Result ☐

Polling Configuration

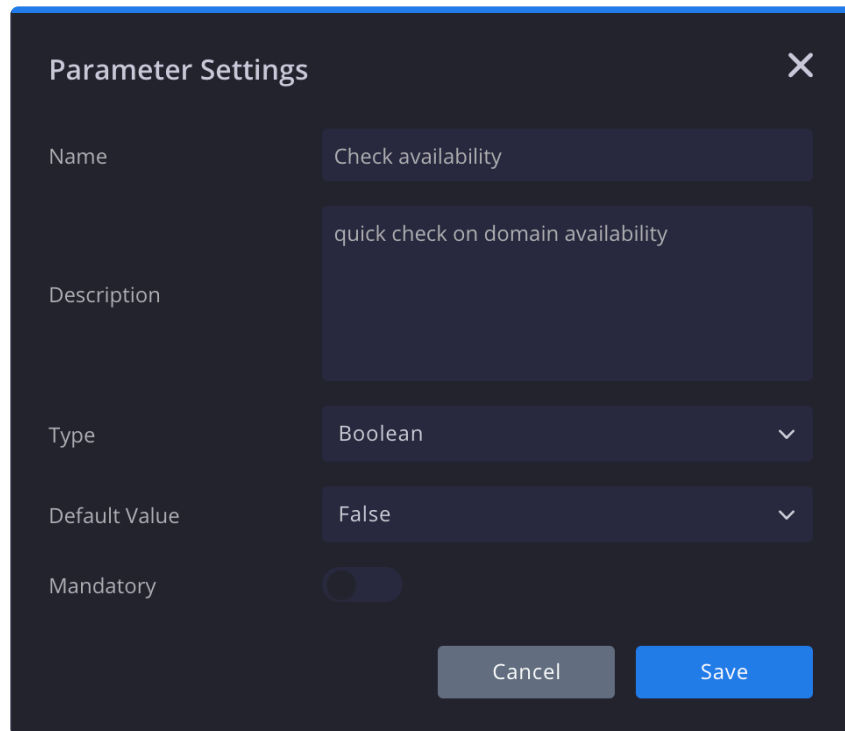
Polling Timeout 0 0 5 0
Days Hours Minutes Seconds

Default Return Value

Parameters   

Parameter	Type	Default Value	Mandatory	Description
No records found				

2. Create the first parameter and fill in the fields as presented in the screenshot for the “Check availability” parameter and click on the save button. This parameter indicates if the domain is available or not and the result will be used in the automation we create.

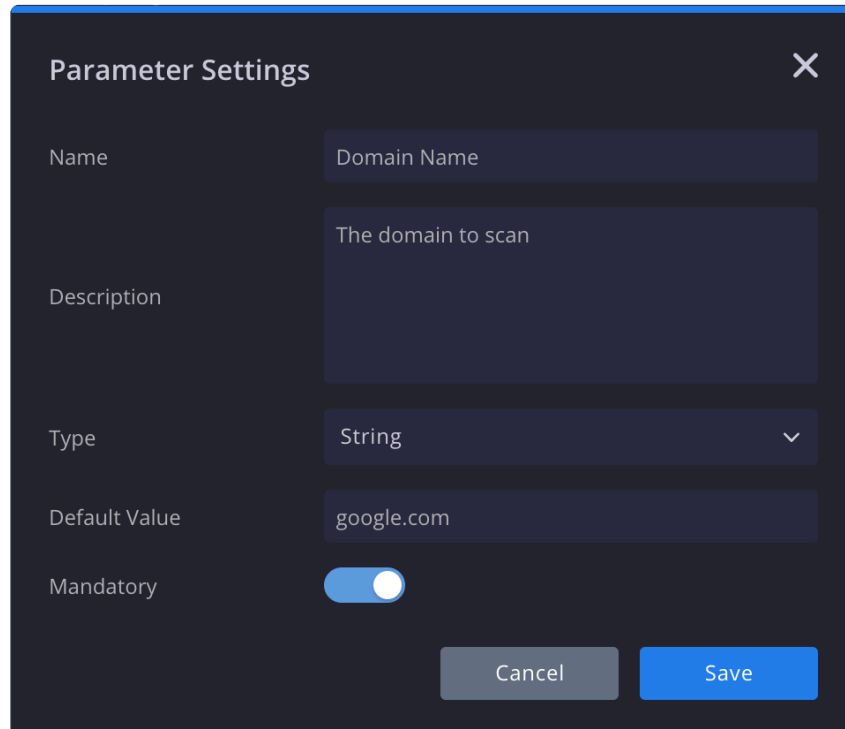


The screenshot shows a 'Parameter Settings' dialog box with a dark theme. It has a title bar with a close button (X). The fields are as follows:

- Name:** Check availability
- Description:** quick check on domain availability
- Type:** Boolean (dropdown menu)
- Default Value:** False (dropdown menu)
- Mandatory:** A toggle switch that is currently turned off.

At the bottom right, there are two buttons: 'Cancel' (grey) and 'Save' (blue).

3. Create the second parameter and fill in the fields as presented in the screenshot for the "Domain Name" parameter and click on the save button. This field will be used to insert the domain name we would like the action to check its details.



The screenshot shows a 'Parameter Settings' dialog box with a dark theme. It has a title bar with a close button (X). The fields are as follows:

- Name:** Domain Name
- Description:** The domain to scan
- Type:** String (dropdown menu)
- Default Value:** google.com
- Mandatory:** A toggle switch that is currently turned on.

At the bottom right, there are two buttons: 'Cancel' (grey) and 'Save' (blue).

Edit the Get Domain Details Action

1. Copy the below code created for the Get Domain Details, paste it in the IDE and follow the explanation of the action.

```
from SiemplifyAction import SiemplifyAction
from SiemplifyUtils import output_handler
import requests

# Example Consts:
INTEGRATION_NAME = "My first Integration - Whois XML API"

SCRIPT_NAME = "WHOIS XML API GetDomainDetails"

@output_handler
def main():
    siemplify = SiemplifyAction()
    siemplify.script_name = SCRIPT_NAME
    siemplify.LOGGER.info("===== Main - Param Init
=====")

    api_key = siemplify.extract_configuration_param(provider_name=INTEGRATION_NAME,
                                                    param_name="API Key")

    url = f"https://www.whoisxmlapi.com/whoisserver/WhoisService?apiKey={api_key}&outputFormat=json"

    domain = siemplify.extract_action_param(param_name="Domain Name", print_value=True)
    availabilty_check = siemplify.extract_action_param(param_name="Check availability", is_mandatory=False, print_value=True)

    # Add domain to scan
    url = f"{url}&domainName={domain}"

    # Determine availabilty check
    if availabilty_check.lower() == 'true':
        availabilty_check_qs = 1
    else:
        availabilty_check_qs = 0
```

```

url = f"{url}&da={availability_check_qs}"

response = requests.get(url)
response.raise_for_status()

# Add a Json result that can be used in the next steps of the playbook.
simplify.result.add_result_json(response.json())
# Add the Json to the action result presented in the context details.
simplify.result.add_json("WhoisDetails", response.json())

msg = f"Fetch data for {domain}"

simplify.end(msg, None)

if __name__ == "__main__":
    main()

```

There are two things that must happen in a Simplify Action. Firstly, an object must be instantiated from the `SimplifyAction` class that extracts the Simplify SDK.

```

from SimplifyAction import SimplifyAction
from SimplifyUtils import output_handler
import requests

```

The second is that the object must utilize the class's `end` method to return an output message and a result value.

```
simplify.end(msg, None)
```

2. Extract integration & action params – as you can see in the code copied into the action, from line 17 to 24 we use the `simplify.extract_configuration_param` function which extracts the parameters configured for the integration (**API Key**) and `simplify.extract_action_param` function which extracts each of the parameters we configured for the action (**Domain Name & Check availability**).

```

api_key = simplify.extract_configuration_param(provider_name=INTEGRATION_NAME,
                                              param_name="API Key")

url = f"https://www.whoisxmlapi.com/whoisserver/WhoisService?apiKey={api_key}"

```

```
y}&outputFormat=json"
```

```
domain = simplify.extract_action_param(param_name="Domain Name", print_value=True)
availability_check = simplify.extract_action_param(param_name="Check availability", is_mandatory=False, print_value=True)
```

3. Once we have extracted the parameters configured for the integration and Action we will then build the URL according to the boolean `availability_check`. Once the url is ready, we will create a request to the whois site, parse the answer and will add it to the result of the action. We then add the Json result and define the output message that will be presented in the action result.

```
# Add domain to scan
url = f"{url}&domainName={domain}"

# Determine availability check
if availability_check.lower() == 'true':
    availability_check_qs = 1
else:
    availability_check_qs = 0

url = f"{url}&da={availability_check_qs}"

response = requests.get(url)
response.raise_for_status()

# Add a Json result that can be used in the next steps of the playbook.
simplify.result.add_result_json(response.json())
# Add the Json to the action result presented in the context details.
simplify.result.add_json("WhoisDetails", response.json())

msg = f"Fetch data for {domain}"

simplify.end(msg, None)

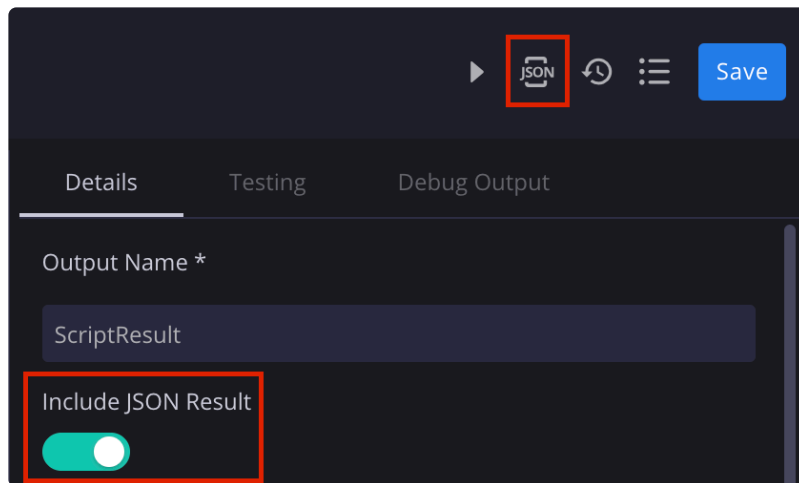
if __name__ == "__main__":
    main()
```

Adding a JSON Result to the Action

As part of the Get Domain Details action we will also add a Json example to the action using the “Include

JSON Result” toggle. We will be utilizing the Json example in the playbook designer in the “[My First Automation](#)” tutorial to extract a specific field in the Json.

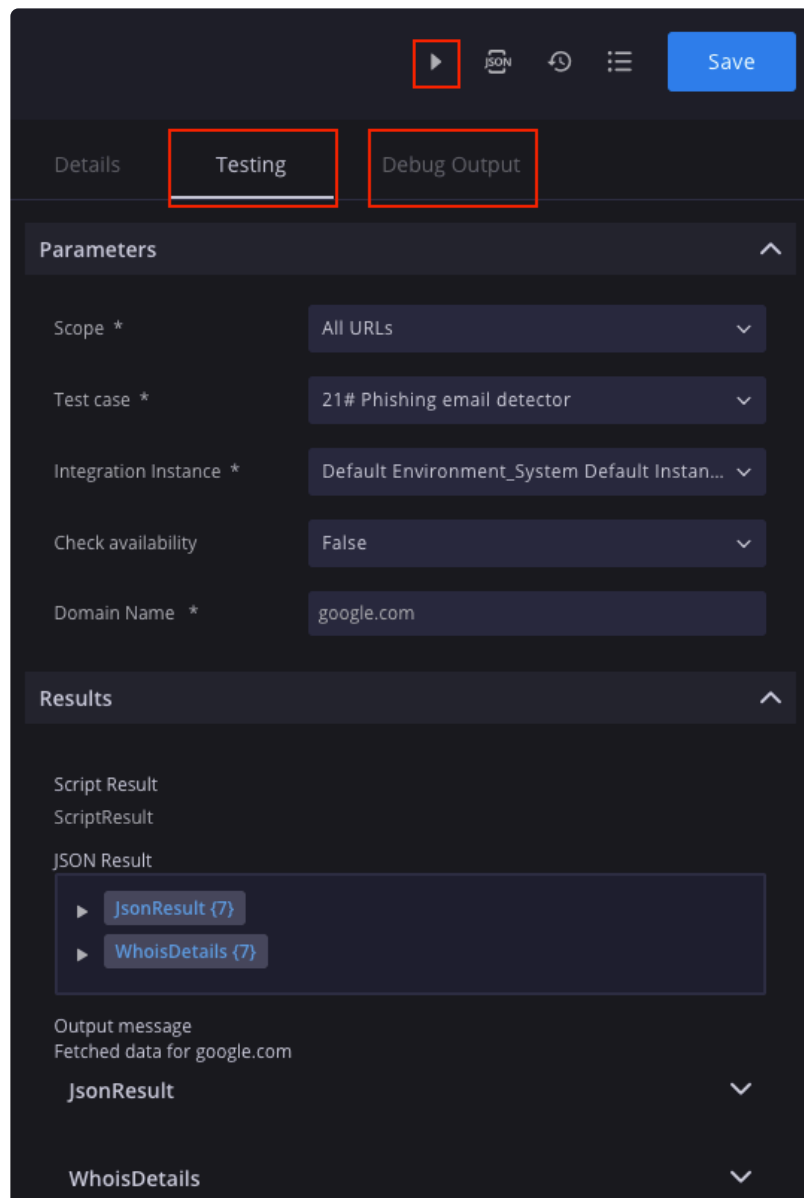
1. In order to insert a JSON example copy the JSON from the Example in the WHOIS site as presented in the following [link](#).
2. Enable the toggle in the “Details” tab in the IDE that will enable the JSON icon in the top part of the IDE. Click on the JSON icon and import the JSON from the example in the previous link.



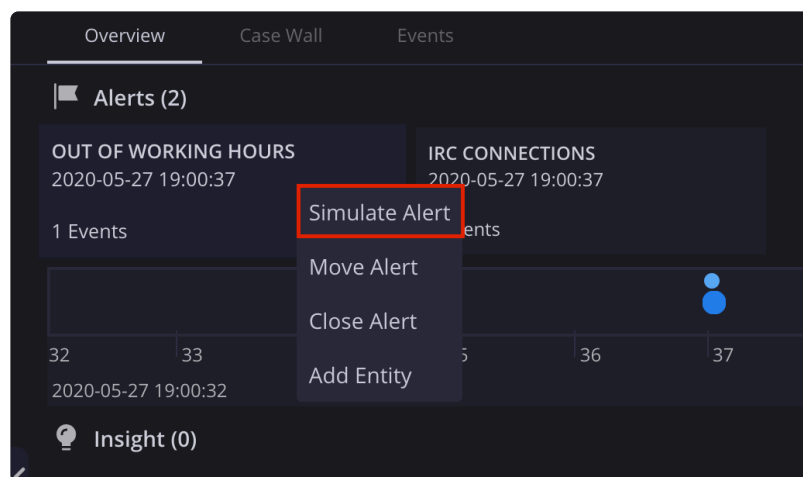
Testing the Action

Once we have finished creating the action we will test the action on a test case.

1. Navigate to the “Testing” tab and choose the Scope, Test Case and Integration Instance.
2. Once all the fields are filled click on the play icon in the top part of the IDE and view the result of the action in the Testing tab. You can also view the Debug output once that test has been completed by navigating to the “Debug output” tab. Please note that the debug shows prints and logs.



If you don't have any Test Cases in your environment, navigate to the cases module and click on simulate alert in one of your cases. This action will create a test case that will be presented with a "Test" label in your case queue. Once you have finished creating the test case navigate back to the IDE and choose the test case from the list.



Creating an Enrichment action

1. Part of the automation we will create in the next how to [“My First Automation”](#) will include an enrichment action that will enrich the entities and add the enrichment data to the different entities which can be viewed in the Entity Explorer.
2. Start off by creating a new action in the IDE and provide it the name “Enrich Entities”. Copy the below code:

```
from SiemplifyAction import SiemplifyAction
from SiemplifyUtils import output_handler
from SiemplifyDataModel import EntityTypes

import requests

# Example Consts:
INTEGRATION_NAME = "My first Integration - Whois XML API"

SCRIPT_NAME = "WHOIS XML API EnrichEntities"

@output_handler
def main():
    siemplify = SiemplifyAction()
    siemplify.script_name = SCRIPT_NAME
    siemplify.LOGGER.info("===== Main - Param Init
=====")

    api_key = siemplify.extract_configuration_param(provider_name=INTEGRATION_NAME,
                                                    param_name="API Key")
```



```

url = f"https://www.whoisxmlapi.com/whoisserver/WhoisService?apiKey={api_key}&outputFormat=json"

simplify.LOGGER.info("----- Main - Started -----")

output_message = "output message :" # human readable message, showed in UI as the action result
successfull_entities = [] # In case this actions contains entity based logic, collect successfull entity.identifiers

for entity in simplify.target_entities:
    simplify.LOGGER.info(f"processing entity {entity.identifier}")
    if (entity.entity_type == EntityTypes.HOSTNAME and not entity.is_internal) or entity.entity_type == EntityTypes.URL:
        entity_to_scan = entity.identifier

        scan_url = f"{url}&domainName={entity_to_scan}"

        response = requests.get(scan_url)
        response.raise_for_status()
        register_details = response.json().get("WhoisRecord", {}).get("register_details", {})

        if register_details:
            entity.additional_properties.update(register_details)
            successfull_entities.append(entity)

if successfull_entities:
    output_message += "\n Successfully processed entities:\n    {}".format("\n    ".join([x.identifier for x in successfull_entities]))
    simplify.update_entities(successfull_entities) # This is the actual enrichment (this function sends the data back to the server)
else:
    output_message += "\n No entities where processed."

result_value = len(successfull_entities)

simplify.LOGGER.info("----- Main - Finished -----")
simplify.end(output_message, result_value)

```

```
if __name__ == "__main__":
    main()
```

3. As seen in the code and done previously in the “Get Domain Details” action, we extract the parameters of the integration – the WHOIS XML API Key.
4. We are using `siemplify.target_entities` in the action which returns a list of all the target entities. We then define the Entity type we would like the action to run on, in this case a non internal Hostname or urls.

```
for entity in siemplify.target_entities:
    siemplify.LOGGER.info(f"processing entity {entity.identifier}")
    if (entity.entity_type == EntityTypes.HOSTNAME and not entity.is_internal) or entity.entity_type == EntityTypes.URL:
        entity_to_scan = entity.identifier
```

5. We then scan the domain, define the enrichment step of the action and the output message. This action runs on an Entity scope and therefore does not require to configure specific parameters, this is already embedded in the code.

```
scan_url = f"{url}&domainName={entity_to_scan}"

response = requests.get(scan_url)
response.raise_for_status()
register_details = response.json().get("WhoisRecord", {}).get("registrant", {})

if register_details:
    entity.additional_properties.update(register_details)
    successfull_entities.append(entity)

if successfull_entities:
    output_message += "\n Successfully processed entities:\n    {}".format(
        "\n    ".join([x.identifier for x in successfull_entities]))
    siemplify.update_entities(successfull_entities) # This is the actual enrichment (this function sends the data back to the server)
else:
    output_message += "\n No entities where processed."
```

```
result_value = len(successfull_entities)
```

6. Enable the action and save it. You now have a custom Integration that you created that has 3 custom actions – a ping action that enabled us to test the connection to the WHOIS XML API product, a Get Domain Details action that extracts data regarding the domain presenting a Json result and a final action that enriches the entities and adds additional data to the target entities that are presented in the Entity Explorer module. Everything is now ready for you to create your first automation using the actions you customized.

1.5. My First Connector

What are Connectors?

The connectors are the entry point for alerts into Siemplify. Their job is to translate raw input data from multiple sources into Siemplify data. The connectors get alerts (or equivalent data – e.g. alarms, correlation events, etc) from 3rd party tools sent to the Data Processing layer, to be ingested as Siemplify alerts and events.

Overview

In this guide, we will demonstrate how to develop an email connector in Siemplify IDE in order to ingest raw data from an email source (Gmail) and translates it into Siemplify data in order to create cases in the Siemplify platform.

The connector will scan each email message body in order to extract URLs from the email. In the next step we will check if these URLs are malicious using the product we have integrated with in [My First Action – Who Is XML](#).



<https://www.youtube.com/embed/Fu8qdQDyUyo?rel=0>


Prerequisite steps

To allow the connector to connect to your email inbox there are a few steps that need to be done.

1. Let's start off by creating a new gmail account or using one that you already have for testing purposes.
2. "2-step verification", is one of the security adjustments to allow Siimplify platform to access the email inbox securely.

First option – turn off the "2-step verification" under "Signing in to google", and turn on the "Less secure app access".

Signing in to Google



Password	Last changed Sep 2	>
Use your phone to sign in	Off	>
2-Step Verification	Off	>

Less secure app access

Your account is vulnerable because you allow apps and devices that use less secure sign-in technology to access your account. To keep your account secure, Google will automatically turn this setting OFF if it's not being used.


On

[Turn off access \(recommended\)](#)

Second option – in order to leave your 2-step verification on, you can create an App Password that gives the Siimplify platform permission to access your Google Account. App Passwords can only be used with accounts that have 2-step verification turned on.

Click on the App passwords icon and then fill in the relevant fields:

Signing in to Google



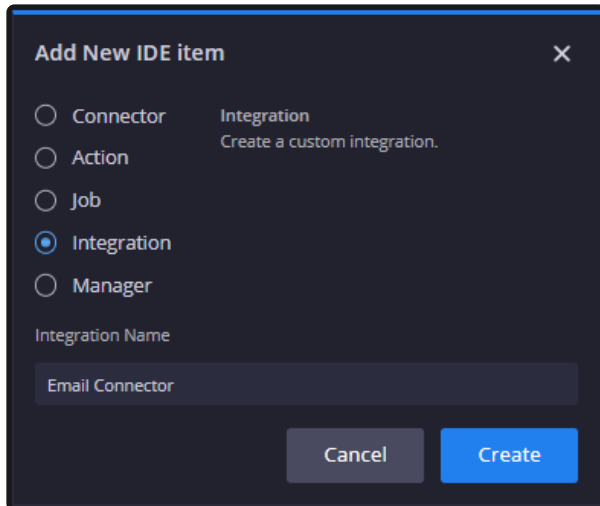
Password	Last changed Aug 10	>
2-Step Verification	On	>
App passwords	None	>

“Select app”: select “Other (Custom name)” option and add URL associated with your Simplify platform (DNS).

The next step is to create the email connector in the IDE. Continue to the [Developing the Connector](#) section.

1.5.1. Developing the Connector

1. From the IDE screen click the + icon in the upper left hand corner to add a new IDE item. Select the Integration radio button and give the integration the name: "Email Connector".



Add New IDE item [X]

☐ Connector Integration
Create a custom integration.

☐ Action

☐ Job

☒ Integration

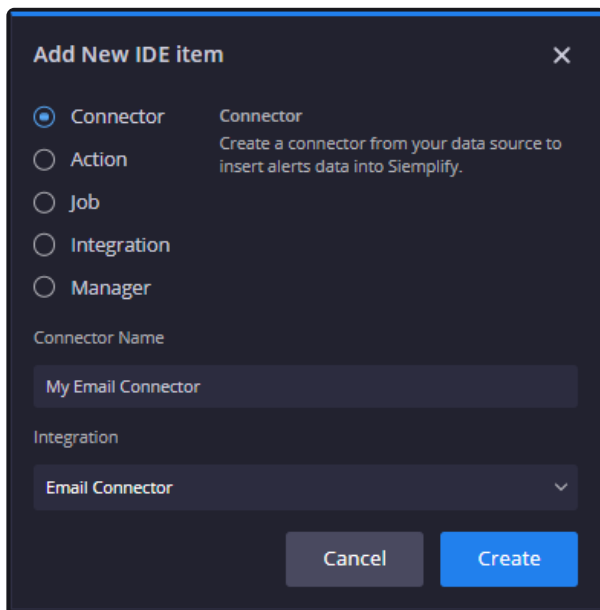
☐ Manager

Integration Name

Email Connector

Cancel Create

2. The integration will be created and listed on the left hand side with a default icon. Clicking on the Gear Icon will bring up the Integration Settings where the Icon, Description, Python Dependencies and Integration Parameters can be defined.
3. Next, click the + icon and add a new IDE item. Select the Connector radio button and give the connector the name: "My Email Connector". Next, select the integration "Email Connector" to associate the connector with the integration.



Add New IDE item [X]

☒ Connector Connector
Create a connector from your data source to insert alerts data into Siemplify.

☐ Action

☐ Job

☐ Integration

☐ Manager

Connector Name

My Email Connector

Integration

Email Connector

Cancel Create

4. After creating the connector, set the following connector parameters:

* Note: in this example the connector pulls only the unread messages and after processing each email message it will be automatically marked as read in the email box.

Parameter Name	Description	Mandatory	Type	Default Value	Explanation
Username	IMAP User name	Yes	String	email@gmail.com	The email address from which the connector will pull the emails into Siemplify platform
Password	IMAP Password	Yes	Password		The password associated with the email address from which the connector will ingest the emails into Siemplify platform
IMAP Port	Imap port. e.g. 993	Yes	Int	993	The Internet Message Access Protocol (IMAP) is a mail protocol used for accessing emails on a remote web server from a local client.
IMAP Server Address	e.g. imap.gmail.com	Yes	String	imap.google.com	The incoming mail server for an IMAP account can also be called the IMAP server. In this example, the email provider is google.com, and the incoming mail server is imap.google.com.
Folder to check for emails	Pulls emails only from the specified folder	No	String	Inbox	The folder from which the emails will be retrieved, For example: Inbox

5. Next, in the upper right fill out the fields:

Details
Whitelist
Testing
Debug Output

Product Field Name *
device_product

Event Field Name *
event_name

Timeout configuration

Script Timeout *
0
0
0
30
Days
Hours
Minutes
Seconds

- “Product Field Name” = device_product, determines which value from the raw fields would be assigned to the product name of the alert. You can find the related field in the code in line 57 which

was defined as "Mail" (product).

```
event["device_product"] = PRODUCT #The PRODUCT constant is "Mail"
```

- "Event Field Name" = event_name, determines which value from the raw fields would be assigned to the event type field. You can find the related field in the code in line 56 which was defined as "Suspicious email".

```
event["event_name"] = "Suspicious email"
```

Edit the Email Connector

1. Copy the code below created for the "My Email Connector", paste it in the IDE and follow the instructions.

```
from SiemplifyConnectors import SiemplifyConnectorExecution
from SiemplifyConnectorsDataModel import AlertInfo
from SiemplifyUtils import output_handler, convert_datetime_to_unix_time, convert_string_to_datetime
import email, imaplib, sys, re

# CONSTANTS
CONNECTOR_NAME = "Mail"
VENDOR = "Mail"
PRODUCT = "Mail"
DEFAULT_PRIORITY = 60 # Default is Medium
RULE_GENERATOR_EXAMPLE = "Mail"
DEFAULT_FOLDER_TO_CHECK_INBOX = "inbox"
DEFAULT_MESSAGES_TO_READ_UNSEEN = "UNSEEN"
URLS_REGEX = r"(?i)\b(?:http(?:s)?://\/)?(?:www\.)?[a-zA-Z0-9:_%\+~#=[a-zA-Z0-9:_%\+~#={1,255}\.[a-z]{2,6}\b(?:[-a-zA-Z0-9@:_%\+~#?&/=]*)"
```

```
def create_alert(siemplify, alert_id, email_message_data, datetime_in_unix_time, created_event):
    """
    Returns an alert which is one event that contains one unread email message
    """
    siemplify.LOGGER.info(f"----- Started processing Alert {alert_id}")
    create_event = None
    alert_info = AlertInfo()

    # Initializes the alert_info Characteristics Fields
    alert_info.display_id = f"{alert_id}" # Each alert needs to have a unique id, otherwise it won't create a case with the same alert id.
```

```

    alert_info.ticket_id = f"{alert_id}" # In default, ticket_id = display_id. However, if for some reason the external alert id is different from the display_id, you can save the original external alert id in the "ticket_id" field.
    alert_info.name = email_message_data['Subject']
    alert_info.rule_generator = RULE_GENERATOR_EXAMPLE # The name of the siem rule which causes the creation of the alert.
    alert_info.start_time = datetime_in_unix_time # Time should be saved in UnixTime. You may use SimplifyUtils.convert_datetime_to_unix_time, or SimplifyUtils.convert_string_to_datetime
    alert_info.end_time = datetime_in_unix_time # Time should be saved in UnixTime. You may use SimplifyUtils.convert_datetime_to_unix_time, or SimplifyUtils.convert_string_to_datetime
    alert_info.priority = 60 # Informative = -1, Low = 40, Medium = 60, High = 80, Critical = 100.
    alert_info.device_vendor = VENDOR # The field will be fetched from the Original Alert. If you build this alert manually, state the source vendor of the data. (ie: Microsoft, McAfee)
    alert_info.device_product = PRODUCT # The field will be fetched from the Original Alert. If you build this alert manually, state the source product of the data. (ie: ActiveDirectory, AntiVirus)
    # ----- Alert Fields initialization END
    -----#
    simplify.LOGGER.info(f"----- Events creating started for alert {alert_id}")
    try:
        if created_event is not None:
            alert_info.events.append(created_event)
            simplify.LOGGER.info(f"Added Event {alert_id} to Alert {alert_id}")
        # Raise an exception if failed to process the event
    except Exception as e:
        simplify.LOGGER.error(f"Failed to process event {alert_id}")
        simplify.LOGGER.exception(e)

    return alert_info

def create_event(simplify, alert_id, email_message_data, all_found_url_in_emails_body_list, datetime_in_unix_time):
    """
    Returns the digested data of a single unread email
    """
    simplify.LOGGER.info(f"--- Started processing Event: alert_id: {alert_id} | event_id: {alert_id}")
    event = {}
    event["StartTime"] = datetime_in_unix_time # Time should be saved in UnixTime

```

```

e. You may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.co
nvert_string_to_datetime
    event["EndTime"] = datetime_in_unix_time # Time should be saved in UnixTime.
You may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.conve
rt_string_to_datetime
    event["event_name"] = "Suspicious email"
    event["device_product"] = PRODUCT # ie: "device_product" is the field name th
at describes the product the event originated from.

    event["Subject"] = email_message_data["Subject"]
    event["SourceUserName"] = email_message_data["From"]
    event["DestinationUserName"] = email_message_data["To"]
    event["found_url"] = ", ".join(all_found_url_in_emails_body_list)

    siemplify.LOGGER.info(f"--- Finished processing Event: alert_id: {alert_id}
| event_id: {alert_id}")
    return event

def find_url_in_email_message_body(siemplify, email_messages_data_list):
    """
    Search for a url in the email body,
    """
    all_found_url_in_emails_body_list = []
    for message in email_messages_data_list:
        for part in message.walk():
            if part.get_content_maintype() == 'text/plain':
                continue
            email_message_body = part.get_payload()
            all_found_urls = re.findall(URLS_REGEX, str(email_message_body))
            for url in all_found_urls:
                if url not in all_found_url_in_emails_body_list:
                    all_found_url_in_emails_body_list.append(url)

    siemplify.LOGGER.info(f"The URL found : {all_found_url_in_emails_body_list}")

    return all_found_url_in_emails_body_list

def get_email_messages_data(imap_host, imap_port, username, password, folder_to_c
heck):
    """
    Returns all unread email messages
    """
    email_messages_data_list = []

```

```

# Login to email using 'imap' module
mail = imaplib.IMAP4_SSL(imap_host, imap_port)
mail.login(username, password)
# Determining the default email folder to pull emails from - 'inbox'
if folder_to_check is None:
    folder_to_check = DEFAULT_FOLDER_TO_CHECK_INBOX
# Selecting the email folder to pull the data from
mail.select(folder_to_check)
# Storing the email message data
result, data = mail.search(None, DEFAULT_MESSAGES_TO_READ_UNSEEN)
# If there are several emails collected in the cycle it will split each email message into a separate item in the list chosen_mailbox_items_list
if len(data) > 0:
    chosen_mailbox_items_list = data[0].split()
    # Iterating each email message and appending to emails_messages_data_list
    for item in chosen_mailbox_items_list:
        typ, email_data = mail.fetch(item, '(RFC822)')
        # Decoding from binary string to string
        raw_email = email_data[0][1].decode("utf-8")
        # Turning the email data into an email object
        email_message = email.message_from_string(raw_email)
        # Appending the email message data to email_messages_data_list
        email_messages_data_list.append(email_message)
    return email_messages_data_list

@output_handler
def main(is_test_run):
    alerts = [] # The main output of each connector run that contains the alerts data
    siemplyfy = SiemplyfyConnectorExecution() # Siemplyfy main SDK wrapper
    siemplyfy.script_name = CONNECTOR_NAME

    #In case of running a test
    if (is_test_run):
        siemplyfy.LOGGER.info("This is an \"IDE Play Button\"\\\"\\\"Run Connector on ce\" test run")

    #Extracting the connector's Params
    username = siemplyfy.extract_connector_param(param_name="Username")
    password = siemplyfy.extract_connector_param(param_name="Password")
    imap_host = siemplyfy.extract_connector_param(param_name="IMAP Server Address")
    imap_port = siemplyfy.extract_connector_param(param_name="IMAP Port")
    folder_to_check = siemplyfy.extract_connector_param(param_name="Folder to che

```

```
ck for emails")

#Getting the digested email message data
email_messages_data_list = get_email_messages_data(imap_host, imap_port, user
name, password, folder_to_check)

#If the email_messages_data_list is not empty
if len(email_messages_data_list) > 0:

    for message in email_messages_data_list:
        # Converting the email message datetime from string to unix time by S
iemplifyUtils functions:
        datetime_email_message = message['Date']
        string_to_datetime = convert_string_to_datetime(datetime_email_messag
e)

        datetime_in_unix_time = convert_datetime_to_unix_time(string_to_datet
ime)

        found_urls_in_email_body = find_url_in_email_message_body(siemplify,
email_messages_data_list)

        # Getting the unique id of each email message and removing the suffi
x '@mail.gmail.com' from the Message-ID, Each alert id can be ingested to the sys
tem only once.
        alert_id = message['Message-ID'].replace('@mail.gmail.com','')

        # Creating the event by calling create_event() function
        created_event = create_event(siemplify, alert_id, message, found_url
s_in_email_body, datetime_in_unix_time)
        # Creating the alert by calling create_alert() function
        created_alert = create_alert(siemplify, alert_id, message, datetime_i
n_unix_time, created_event)

        # Checking that the created_alert is not None
        if created_alert is not None:
            alerts.append(created_alert)
            siemplify.LOGGER.info(f"Added Alert {alert_id} to package result
s")

# If the inbox for the user has no unread emails.
else:
    siemplify.LOGGER.info(f"The inbox for user {username} has no unread emai
ls")
```

```
# Returning all the created alerts to the cases module in Siemplify
siemplify.return_package(alerts)

if __name__ == "__main__":
    # Connectors run in iterations. The interval is configurable from the Connect
    orsScreen UI.
    is_test_run = not (len(sys.argv) < 2 or sys.argv[1] == 'True')
    main(is_test_run)
```

2. Now that we have copied the connectors code we will go over the relevant modules that need to be imported and continue with the main function. Afterwards we will elaborate on each method that was called from the main function.

The relevant imports

A Python module has a set of functions, classes or variables defined and implemented. In order to achieve all the functions below we imported those modules into our script.

```
from SiemplifyConnectors import SiemplifyConnectorExecution # This module is resp
onsible for executing the connector
from SiemplifyConnectorsDataModel import AlertInfo #The data model that contains
the alert info class
from SiemplifyUtils import output_handler, convert_datetime_to_unix_time, conver
t_string_to_datetime #The functions that convert time
import email, imaplib, sys, re
```

Main function

The main function is the start point of the script. The Python interpreter executes the code sequentially and calls each method that is part of the code.

1. Extract connector params – as you can see in the code copied into the IDE, we use the `siemplify.extract_connector_param` function which extracts each of the parameters we configured for the connector (username, password, imap_host, imap_port, folder_to_check).

```
#Extracting the connector's Params
username = siemplify.extract_connector_param(param_name="Username")
password = siemplify.extract_connector_param(param_name="Password")
imap_host = siemplify.extract_connector_param(param_name="IMAP Server Address")
imap_port = siemplify.extract_connector_param(param_name="IMAP Port")
folder_to_check = siemplify.extract_connector_param(param_name="Folder to check f
```

```
or emails")
```

2. We will use the function `get_email_messages_data(imap_host, imap_port, username, password, folder_to_check)` in order to get all the information collected from the unread emails (We will elaborate on this function in another step).

```
#Getting the digested email message data
email_messages_data_list = get_email_messages_data(imap_host, imap_port, username, password, folder_to_check)
```

3. After we have received all the information of the email we will check that the information has indeed been collected, and then we will perform a number of actions on each email:

```
#If the email_messages_data_list is not empty
if len(email_messages_data_list) > 0:
    for message in email_messages_data_list:
        # Converting the email message datetime from string to unix time by SimplifyUtils functions
```

- This code extracts the message date by `datetime_email_message = message['Date']` and then converts this date time to unix time using Simplify functions:

```
string_to_datetime = convert_string_to_datetime(datetime_email_message)
datetime_in_unix_time = convert_datetime_to_unix_time(string_to_datetime)
```

- We then search for URLs (if the email has a url we will use other products in our playbook to check if the URL is malicious) in the email message body by using the function below `find_url_in_email_message_body(simplify, email_messages_data_list)` (We will elaborate on this function in another step).

```
found_urls_in_email_body = find_url_in_email_message_body(simplify, email_messages_data_list)
```

- Extract the unique ID of each email message, and assign it to the `alert_id` variable.

```
# Getting the unique id of each email message and removing the suffix '@mail.gmail.com' from the Message-ID, Each alert id can be ingested to the system only once.
alert_id = message['Message-ID'].replace('@mail.gmail.com', '').
```

- After we extracted all the necessary information for ingesting the alert into the Siemplify platform, we can create the alert and the event (We will elaborate on these functions in another step):

```
# Creating the event by calling create_event() function
created_event = create_event(siemplify, alert_id, message, found_urls_in_email_body, datetime_in_unix_time)
# Creating the alert by calling create_alert() function
created_alert = create_alert(siemplify, alert_id, message, datetime_in_unix_time, created_event)
```

- Next we will validate the created alert and the created event. After validating we will add the alert to the alert list.

```
# Checking that the created_alert is not None
if created_alert is not None:
    alerts.append(created_alert)
    siemplify.LOGGER.info(f"Added Alert {alert_id} to package results")
```

4. In a situation that the inbox for the given user has no unread emails we have added the following code:

```
else:
    siemplify.LOGGER.info(f"The inbox for user {username} has no unread emails")
```

5. At the end we will return the alerts list to the system and each alert will be presented as a case in the case queue.

```
# Returning all the created alerts to the cases module in Siemplify
siemplify.return_package(alerts)
```

6. This step is responsible for running the Main function within the times we set in the Connector configuration:

```
if __name__ == "__main__":
    # Connectors run in iterations. The interval is configurable from the ConnectorsScreen UI.
    is_test_run = not (len(sys.argv) < 2 or sys.argv[1] == 'True')
    main(is_test_run)
```


Getting the unread email message

This function is responsible for connecting to the email by the “Imap” and “Email” models and retrieving the information of the email message. Finally, the function returns a list containing all the information of all the unread email messages.

1. From the main class we will use the function `get_email_messages_data(imap_host, imap_port, username, password, folder_to_check)`.

```
def get_email_messages_data(imap_host, imap_port, username, password, folder_to_check):  
    """  
    Returns all unread email messages  
    """  
    email_messages_data_list = []
```

2. After that we will connect to the email by using the ‘imap’ module.

```
# Login to email using 'imap' module  
mail = imaplib.IMAP4_SSL(imap_host, imap_port)  
mail.login(username, password)
```

3. We will then determine the folder in the email to check for unread messages. In this example we will extract emails from the inbox’ folder (`DEFAULT_FOLDER_TO_CHECK_INBOX = "inbox"`).

```
# Determining the default email folder to pull emails from - 'inbox'  
if folder_to_check is None:  
    folder_to_check = DEFAULT_FOLDER_TO_CHECK_INBOX  
# Selecting the email folder to pull the data from  
mail.select(folder_to_check)
```

4. We then collect all the unread messages(`DEFAULT_MESSAGES_TO_READ_UNSEEN = “UNSEEN”`, and then convert this data to a list.

```
# Storing the email message data  
result, data = mail.search(None, DEFAULT_MESSAGES_TO_READ_UNSEEN)  
# If there are several emails collected in the cycle it will split each email message into a separate item in the list chosen_mailbox_items_list  
if len(data) > 0:  
    chosen_mailbox_items_list = data[0].split()  
    # Iterating each email message and appending to emails_messages_data_list
```

```

for item in chosen_mailbox_items_list:
    typ, email_data = mail.fetch(item, '(RFC822)')
    # Decoding from binary string to string
    raw_email = email_data[0][1].decode("utf-8")
    # Turning the email data into an email object
    email_message = email.message_from_string(raw_email)
    # Appending the email message data to email_messages_data_list
    email_messages_data_list.append(email_message)
return email_messages_data_list

```

Creating the event

This function is responsible for creating the event by associating each email message component to the event fields respectively.

1. From the main class we will create the event by using the function: `create_event(siemplify, alert_id, email_message_data, all_found_url_in_emails_body_list, datetime_in_unix_time)`

```

def create_event(siemplify, alert_id, email_message_data, all_found_url_in_email
s_body_list, datetime_in_unix_time):
    """
    Returns the digested data of a single unread email
    """
    siemplify.LOGGER.info(f"--- Started processing Event:  alert_id: {alert_id}
| event_id: {alert_id}")

```

2. We will create a dictionary with the event fields while the mandatory fields are: `event["StartTime"]`, `event["EndTime"]`, `event["event_name"]` and `event["device_product"]`.

```

event = {}
event["StartTime"] = datetime_in_unix_time # Time should be saved in UnixTime. Yo
u may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.conver
t_string_to_datetime
event["EndTime"] = datetime_in_unix_time # Time should be saved in UnixTime. You
may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.convert_s
tring_to_datetime
event["event_name"] = "Suspicious email"
event["device_product"] = PRODUCT # ie: "device_product" is the field name that d
escribes the product the event originated from.

```

```

event["Subject"] = email_message_data["Subject"]
event["SourceUserName"] = email_message_data["From"]
event["DestinationUserName"] = email_message_data["To"]
event["found_url"] = ",".join(all_found_url_in_emails_body_list)

siemplify.LOGGER.info(f"--- Finished processing Event: alert_id: {alert_id} | eve
nt_id: {alert_id}")
return event

```

- Each alert contains one or more events. In this example we will demonstrate an alert that contains only one event which is a single email message.

Therefore, after creating the event we will create the alert that contains all the event information.

Creating the alert info and initializing the alert info characteristics fields

This function is responsible for creating the alert, each alert contains one or more events within it. In our case each alert contains one event which is basically one email message.

- From the main class we will create the alert by using the function: `create_alert(siemplify, alert_id, email_message_data, datetime_in_unix_time, created_event)`.

```

def create_alert(siemplify, alert_id, email_message_data, datetime_in_unix_time,
created_event):
    """
    Returns an alert which is one event that contains one unread email message
    """
    siemplify.LOGGER.info(f"----- Started processing Alert {alert_id}")
    create_event = None

```

- Creating the alert info instance and initializing the alert info characteristics fields:

```

# Initializes the alert_info Characteristics Fields
alert_info.display_id = f"{alert_id}" # Each alert needs to have a unique id, oth
erwise it won't create a case with the same alert id.
alert_info.ticket_id = f"{alert_id}" # In default, ticket_id = display_id. Howeve
r, if for some reason the external alert id is different from the display_id, yo
u can save the original external alert id in the "ticket_id" field.
alert_info.name = email_message_data['Subject']
alert_info.rule_generator = RULE_GENERATOR_EXAMPLE # The name of the siem rule wh
ich causes the creation of the alert.
alert_info.start_time = datetime_in_unix_time # Time should be saved in UnixTim

```

```
e. You may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.convert_string_to_datetime
alert_info.end_time = datetime_in_unix_time # Time should be saved in UnixTime. You may use SiemplifyUtils.convert_datetime_to_unix_time, or SiemplifyUtils.convert_string_to_datetime
alert_info.priority = 60 # Informative = -1, Low = 40, Medium = 60, High = 80, Critical = 100.
alert_info.device_vendor = VENDOR # The field will be fetched from the Original Alert. If you build this alert manually, state the source vendor of the data. (ie: Microsoft, McAfee)
alert_info.device_product = PRODUCT # The field will be fetched from the Original Alert. If you build this alert manually, state the source product of the data. (ie: ActiveDirectory, AntiVirus)
# ----- Alert Fields initialization END
-----#
```

3. After creating the alert info we will validate the created event and then append the event information to the alert info characteristics.

```
siemplify.LOGGER.info(f"----- Events creating started for alert {alert_id}")
try:
    if created_event is not None:
        alert_info.events.append(created_event)
        siemplify.LOGGER.info(f"Added Event {alert_id} to Alert {alert_id}")
    # Raise an exception if failed to process the event
except Exception as e:
    siemplify.LOGGER.error(f"Failed to process event {alert_id}")
    siemplify.LOGGER.exception(e)

return alert_info
```

Finding the URL in the email body

This function checks if the body of the email has one or more URLs.

For each email message we will need to access the email body by searching the email message part that contains text or plain type information.

```
def find_url_in_email_message_body(siemplify, email_messages_data_list):
    """
    Search for a url in the email body,
```

```
"""
all_found_url_in_emails_body_list = []
for message in email_messages_data_list:
    for part in message.walk():
        if part.get_content_maintype() == 'text/plain':
            continue
```

If the body contains the wanted type of information we will load this information by `email_message_body = part.get_payload()`.

After loading all the information we can now search the URL by using the regex format:

```
URLS_REGEX=r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+"
```

which extracts the URLs from the email body.

```
email_message_body = part.get_payload()
all_found_urls = re.findall(URLS_REGEX, str(email_message_body))
for url in all_found_urls:
    if url not in all_found_url_in_emails_body_list:
        all_found_url_in_emails_body_list.append(url)

simplify.LOGGER.info(f"The URL found : {all_found_url_in_emails_body_list}")

return all_found_url_in_emails_body_list
```

We have finished going through the connector code and we will now configure a connector that will ingest cases into the platform from a selected email inbox in Gmail.

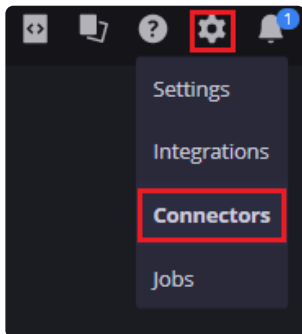
1.5.2. Configuring the Connector

Overview

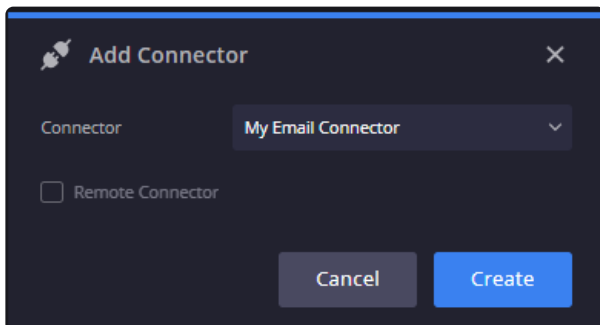
When a new connector is being configured, the platform uses the connector script in an integration as a template only, and the configured connector is an instance of that connector template. You can add multiple connectors with different configurations using the same code you created for the connector in the IDE.

Connector Configuration

1. Select the gear icon in the upper right hand corner to access the connectors module and configure a connector under the relevant environment.



2. Next, from the Connectors screen click the + icon in the upper left hand corner to add a new Connector item.



3. Configure the Connector parameters and select the environment relevant for the connector. In this example, the connector is configured under the Default Environment. Once you fill in all the credentials, save the connector.

My Email Connector

Integration

Email Connector

Definition name

My Email Connector

Description

Parameters

Testing

Logs

Environment *

i

Default Environment

▼

Run Every

0

▲▼

0

▲▼

0

▲▼

10

▲▼

Days

Hours

Minutes

Seconds

Product Field Name *

i

device_product

Event Field Name *

i

event_name

Folder to check for emails *

i

Inbox

Username *

i

testsiemplify2020@gmail.com

Script Timeout (Seconds) *

i

30

Password *

i

.....

IMAP Server Address *

i

imap.gmail.com

IMAP Port *

i

993

In the next step we will test the Connector and ingest a test case into Siemplify platform.

1.5.3. Testing the Connector

In this section we will show an example of an alert that is ingested into the Simplify platform.

1. Insert a malicious email into the platform.

copy the text below and send this email from another user:

Subject: Your New Salary Notification

Email body:

Hello, You have an important email from the Human Resources Department with regards to your December 2018 Paycheck

This email is enclosed in the Marquette University secure network.

Access the documents here <http://markossolomon.com/F1q7QX.php>

Ensure your login credentials are correct to avoid cancellations

Faithfully

Human Resources

University of California, Berkeley

2. Navigate to the 'Testing' tab and test your connector by clicking the button 'Run connector once' and view the result in the "Output" section on the right.

If your connector runs successfully you will see an alert which is a single unread email message that the connector ingested (make sure that you have an unread email in your mailbox to insert a sample alert).

Parameters Testing Logs

Run connector once ☒

Sample Alerts Load to system

Alert Name	Product	Start Time	End Time	Event Count	Preview
<input type="checkbox"/> Your New Salary Not...	Mail	2020-10-19 18:38:44	2020-10-19 18:38:44	1	

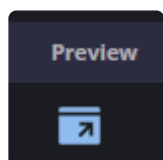
Output

```

***** This is an "IDE Play Button"/"Run Connector once" test run *****
Inbox
The URL found : ["http://markossolomon.com/F1q7QX.php"]
--- Started processing Event: alert_id: <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA> |
event_id: <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA>
--- Finished processing Event: alert_id: <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA> |
event_id: <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA>
----- Started processing Alert <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA>
----- Events creating started for alert: <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA>
Added Event <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA> to Alert
<CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA>
Added Alert <CABVUvqa6rgv7zeXGg9GVz3esekVmme-fauy7VdYuYqZK08GmA> to package results

```

3. You can see a preview of the email by clicking on the preview icon.



4. After ingesting a sample alert by clicking the "Run connector once" we will ingest the alert into the case queue by selecting the alert and clicking the button 'Load to system'.

Sample Alerts						Load to system
<input type="checkbox"/>	Alert Name	Product	Start Time	End Time	Event Count	Preview
<input type="checkbox"/>	Your New Salary Not...	Mail	2020-10-19 18:38:44	2020-10-19 18:38:44	1	

5. Navigate to the Cases tab and view the case you have ingested into the Siemplify platform.

Siemplify | COMMUNITY

HOMEPAGE

DASHBOARDS

CASES

PLAYBOOKS

SEARCH

REPORTS

COMMAND CENTER

5 Cases

Mail

Modified 2020-10-21 17:24:13 ID 280

1

Mail

Modified 2020-10-21 16:36:54 ID 279

1

Mail

Modified 2020-10-21 16:36:54 ID 278

1

Mail

Modified 2020-10-21 16:36:53 ID 277

1

Mail

Modified 2020-10-21 16:32:10 ID 276

1

Overview

Case Wall

Events

Alerts (1)

YOUR NEW SALARY NOT...

Time not mapped

1 Events

Insight (0)

No Insights

Playbooks (0)

No Playbook attached to Alert

Write a comment...

Mail

ID 280

2020-10-21 17:24:13

Default Environment

Triage

Add case description

Simulated Case X

+ Tags

Alert

Your New Salary Notification

Search...

Highlighted Fields

Field Name

Value

Alert Name

YOUR NEW SALA...

DeviceProduct

Mail

DeviceVendor

Mail

Name

Your New Salary ...

End Time

2020-10-21 17:2...

Start Time

2020-10-21 17:2...

Time

Field Name

Value

Detection Time

2020-10-21 17:2...

End Time

2020-10-21 17:2...

Start Time

2020-10-21 17:2...

Estimated Start Ti...

2020-10-21 17:2...

Case

Field Name

Value

Alert Name

YOUR NEW SALA...

Rule Generator

Mail

Ticket ID

3ad9174b-3dfa...

6. After the connector receives the email by translating the email data to Siemplify data we can see our alert in the “Cases” tab in the case queue.

When the case first appears in the system it is not mapped and modeled, this will be done in the next step.

Next, we will see how each field in the code corresponds to the relevant field presented in the context details in the platform itself. Click on the alert to view the Alert Context details on the right.

The field in the platform	The field in the code
<div><div>Case</div><div><div>Field Name</div><div>Value</div><div>Alert Name</div><div>YOUR NEW SALARY NOTIFICATION</div></div></div>	<pre>alert_info.name = email_message_data['Subject']</pre> This represents the subject of the

	email message : "YOUR NEW SALARY NOTIFICATION"				
<div>Rule GeneratorMail⋮</div>	alert_info.rule_generator = RULE_GENERATOR_EXAMPLE # The name of the siem rule which causes the creation of the alert				
<div>Ticket IDa4a0da55-784a-4588-aacc-dbac2450a0f⋮</div>	alert_info.ticket_id = f"{alert_id}" The email message unique id				
<div>Alert IDa4a0da55-784a-4588-aacc-dbac2450a0f⋮</div>	alert_info.display_id = f"{alert_id}" The email message unique id				
<div><div>System^</div><table><thead><tr><th>Field Name</th><th>Value</th></tr></thead><tbody><tr><td>DeviceProduct</td><td>Mail⋮</td></tr></tbody></table></div>	Field Name	Value	DeviceProduct	Mail⋮	alert_info.device_product = PRODUCT As we defined in CONSTANTS: PRODUCT= "Mail"
Field Name	Value				
DeviceProduct	Mail⋮				
<div>DeviceVendorMail⋮</div>	alert_info.device_vendor = VENDOR As we defined in CONSTANTS: VENDOR = "Mail"				

Time

Field Name	Value	
Detection Time	2020-10-12 11:25:36	:
End Time	2020-10-12 11:25:36	:
Start Time	2020-10-12 11:25:36	:
Estimated Start Time	2020-10-12 11:25:36	:

```
alert_info.start_time = datetime_in_unix_time alert_info.end_time = datetime_in_unix_time
```

The time the email message was received

Threat

Field Name	Value	
Priority	Medium	:

```
alert_info.priority = 60
```

As we defined for this alert: Informative = -1, Low = 40, Medium = 60, High = 80, Critical = 100.

1.5.4. Mapping & Modeling

Your alert is not mapped and modeled by default. In order to do so, navigate to the mapping and modelling section (click the gear icon).

Overview Case Wall Events								
Alert #	Alert Name	Event Name	Event Type	Product	Artifacts	Port	Outcome	Time
1	YOUR NEW SALARY NOTIFIC...		Suspicious email	Mail				2020-10-12 11:59:10

- For this use case we will map our case using the predefined family – **MailRelayOrTAP** for email monitoring events.

The screenshot shows the Siemplify Event configuration interface. The top navigation bar includes Home, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, REPORTS, and COMMAND CENTER. The main header shows 'Event configuration' and 'Email Connector > Mail > Suspicious email'. The left sidebar has 'Visualization' and 'Mapping' tabs. The main content area is divided into 'Model families' and 'System Fields'.

Model families

Rule Level	Target field	Extracted Field	Alternative Field 1	Alternative Field 2	Transformation Func...	Transformation Para...
Source	FileName				TO_STRING	
Source	DestinationUse...				TO_STRING	
Source	DestinationPos...				TO_STRING	
Source	SourceUserNa...				TO_STRING	

System Fields

Target Field	Extracted Field	Alternative Field 1	Alternative Field 2	Transformation Function	Transformation Parameters
StartTime				TO_STRING	
EndTime				TO_STRING	
Message				TO_STRING	
Name				TO_STRING	
CategoryOutcome				TO_STRING	
DestinationDnsDomain				TO_STRING	
DestinationNdDomain				TO_STRING	
DestinationPort				TO_STRING	
SourceDnsDomain				TO_STRING	
SourceNdDomain				TO_STRING	

On the right, there is a diagram for the 'MailRelayOrTAP' event, showing a flow from IP to IP through a mail relay, with a checkmark indicating success.

- Mapping and Modeling can be in one of the three stages of hierarchy, for this example:

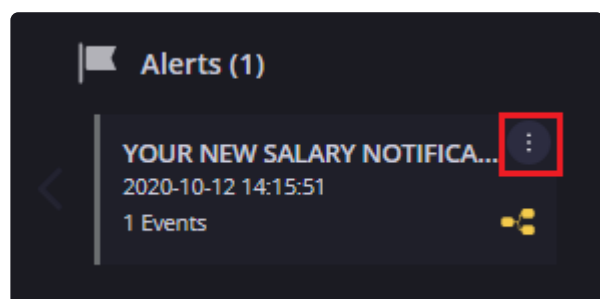
- Source** – This is the Source name field as we filled earlier. This is the Source that digested the data and created an alert in Siemplify platform. For this example the Source name is “Email Connector”. In this stage we will map only the time, since these fields are the same in each stage. If you map at this stage then the following stages (Product – “Mail” and the Event -“Suspicious email”), will inherit the same modeling mapping we performed.
- Product** – The product is “Mail”, which is the product that digests the data that came by the source “Mail”. For example a connector can digest data from many sources. If mapping and modeling is configured at this stage then the following stage (“Suspicious email”) will inherit the same modeling mapping we performed.
- Event** – This is the `event_name` as we filled in earlier, for this example the event name is “Suspicious email”. The event in this case is the email message itself.

3. We will map the relevant fields by assigning each field to the appropriate field in the code. In this mapping section we will map all the fields under the “Product” level.

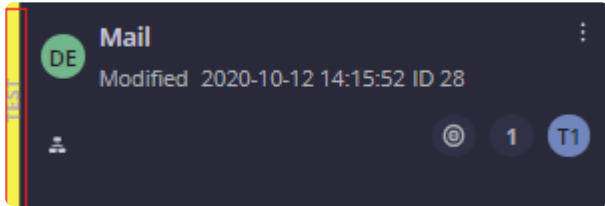
Rule Level	Target Field	Extracted Field	Transformation Function	The field value
Product	DestinationUserName	event["destinationUserName"]	TO_STRING	The email address of the person who received the email.
Product	SourceUserName	event["sourceUserName"]	EXTRACT_BY_REGEX Regex format: <code>[\w\.-]+\@([\w\.-]+)</code>	The email address of the person who sent the email
Product	EmailSubject	event["subject"]	TO_STRING	The email subject
Product	DestinationURL	event["found_url"]	TO_STRING	The URLs found in the email body
Product	StartTime	event["startTime"]	FROM_UNIXTIME_STRING_OR_LONG	The time the email was received
Product	EndTime	event["endTime"]	FROM_UNIXTIME_STRING_OR_LONG	The time the email was received

* Please note that you can click on the information icon to view the transformation function as presented in the pic below.

4. After Mapping this case we will simulate the alert to see the mapping result, on your right hand click on the three dots icon and select “Simulate Alert”.



Then, a new simulated alert will appear as a new case in the case queue. All the simulated cases are tagged with the yellow “Test” mark on the left of the case name.



After mapping the case you can see each email message arguments that we mapped on the right of the screenshot below.

Alerts (1)

YOUR NEW SALARY NOTIFICAT...
2020-10-14 09:42:50
1 Events

2020-10-11 09:42:50

Mon 12 Tue 13 Wed 14 Thu 15 Fri 16 Sat 17

Insight (0)

No Insights

Playbooks (0)

No Playbook attached to Alert

Alert Entities

YOUR NEW SALARY NOTIFICATION
HTTP://MARKOSSOLOMON.COM/F1Q7QX.PHP

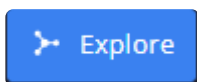
Highlighted Fields

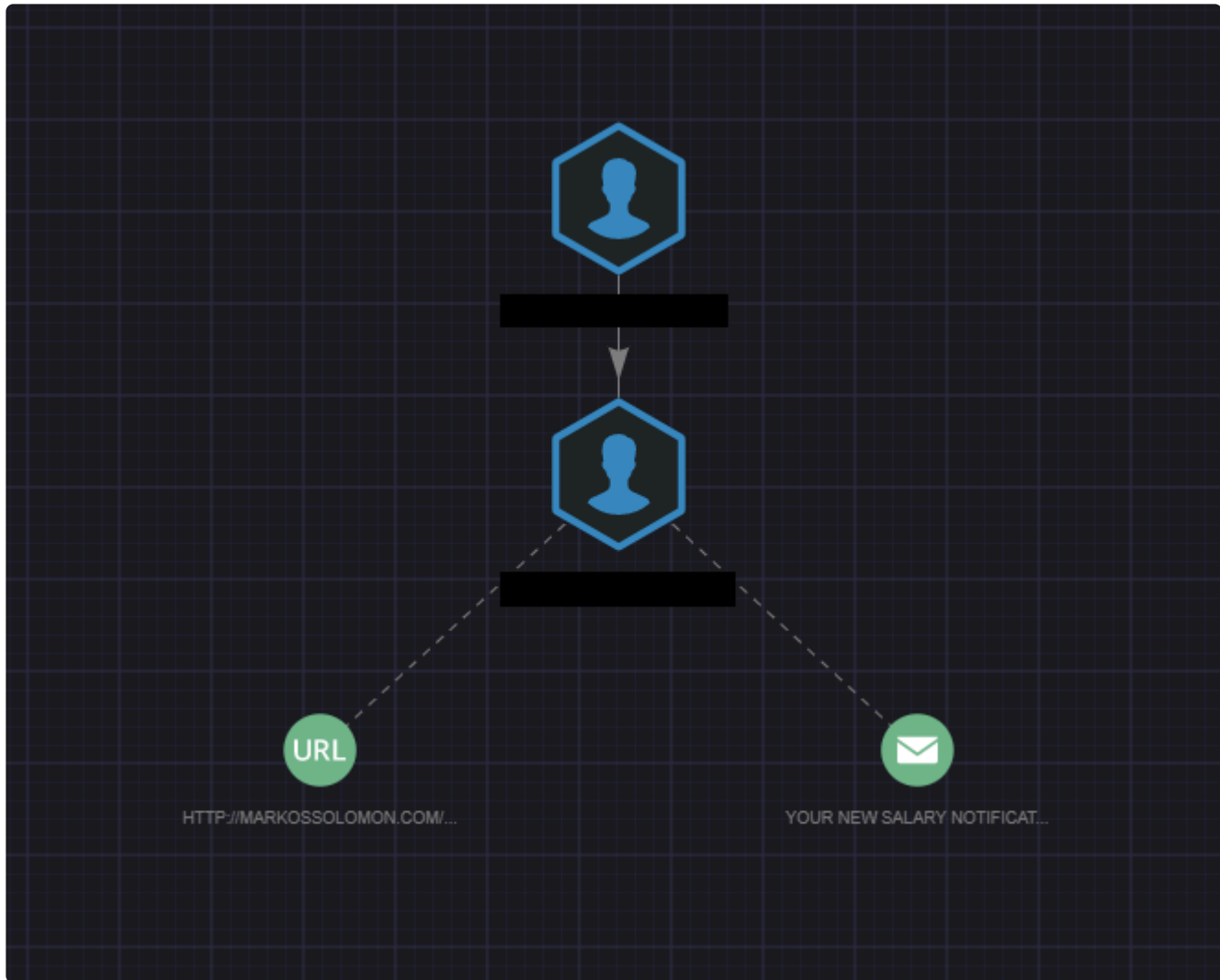
Field Name	Value
Alert Name	YOUR NEW SALARY ...
DeviceProduct	Mail
DeviceVendor	Mail
Name	Your New Salary Not...
End Time	2020-10-14 09:42:50
Start Time	2020-10-14 09:42:50

Time


Field Name	Value
Detection Time	2020-10-14 09:42:50
End Time	2020-10-14 09:42:50
Start Time	2020-10-14 09:42:50
Estimated Start Time	2020-10-14 09:42:50

If you would like to see a visual view of the entities involved in the event and the relations between them, click on the Explore button.





Now that you have finished the mapping and modelling step you can now start ingesting alerts into your platform automatically that will inherit the mapping and modelling you have performed. To do so, navigate back to the Connectors screen, enable the toggle and click save.



My Email Connector
IntegrationEmail Connector
Definition nameMy Email Connector

Description

Parameters

Testing

Logs

Environment *

i

Default Environment

▼

Run Every

0

▲▼

Days

0

▲▼

Hours

0

▲▼

Minutes

10

▲▼

Seconds

Product Field Name *

i

device_product

Event Field Name *

i

event_name

Folder to check for em... *

i

Inbox

Username *

i

testsiimplify2020@gmail.com

Script Timeout (Seconds) *

i

30

Password *

i

.....

IMAP Server Address *

i

imap.gmail.com

IMAP Port *

i

993

Congratulations!! You have developed your first connector in Simplify platform.

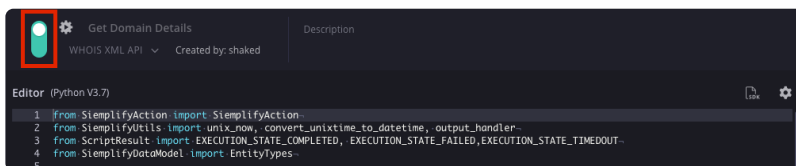
1.6. My First Automation

Overview

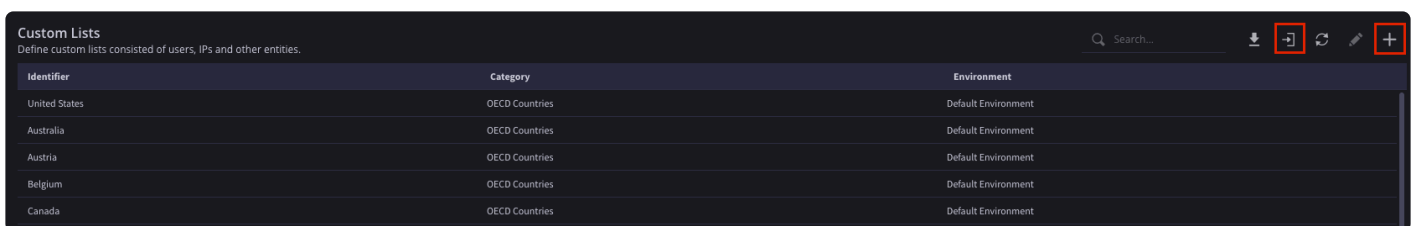
In this article you will learn how to create your first automation using the actions you created in the [My First Action](#). The playbook we will create is a basic phishing use case where we will extract the domain details from the url which is part of an alert. We will then enrich that entity, add an insight with the Domain country, check if the country is in a custom list and then run an IF condition on the check if in custom list to determine if the case requires further investigation or not.

Create your first Playbook

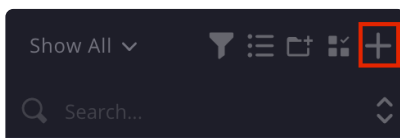
1. In the IDE module make sure that the actions you created as part of the “WHOIS XML API” integration are enabled using the green toggle. Once the actions are enabled, they will only then be available to use in the playbook designer.



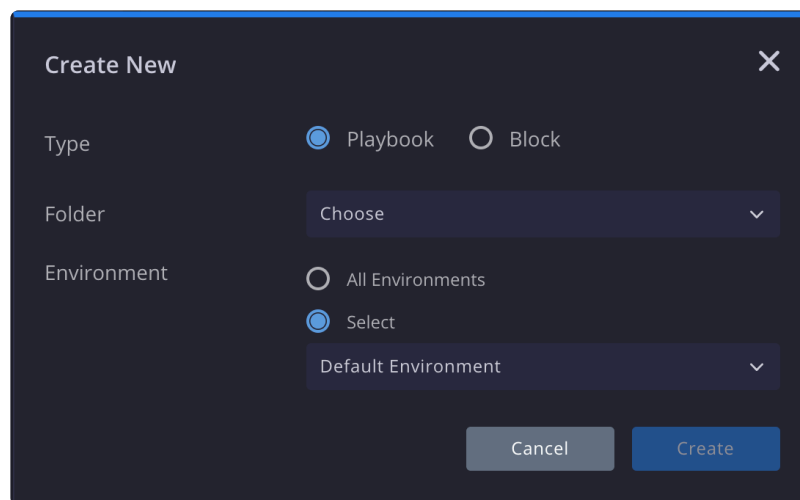
2. For your First Automation Playbook we will also create a custom list of the countries that are OECD countries and use this custom list to determine if the country of the Domain requires further investigation of the case or not. Navigate to the **Settings** module in the top bar, click on the **Environments** tab and then on the **Custom lists**. We have created a custom list of countries that are a [“list of OECD countries”](#) that you can import to your platform using the import icon. You can also customize your own custom list using the plus icon.



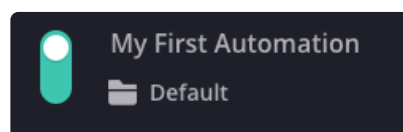
3. To create your First Automation playbook, navigate to the Playbook Designer and click on the + icon in the left part of the playbook queue.



4. In the Create New dialog choose the “Playbook” radio button, select a folder the playbook will be presented in and define the environment.

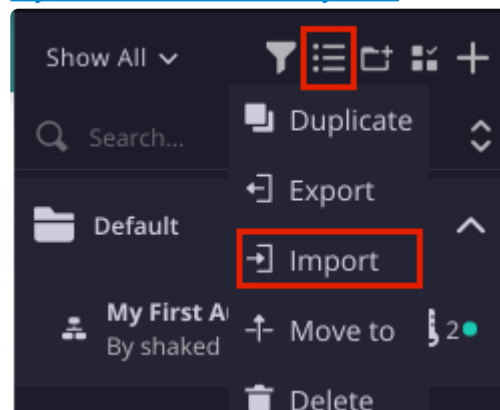
A dark-themed dialog box titled "Create New" with a close button (X) in the top right corner. It contains three sections: "Type" with radio buttons for "Playbook" (selected) and "Block"; "Folder" with a dropdown menu showing "Choose"; and "Environment" with radio buttons for "All Environments" and "Select" (selected), followed by a dropdown menu showing "Default Environment". At the bottom are "Cancel" and "Create" buttons.

5. Provide a name for the playbook next to the playbook toggle and begin to customize the playbook.

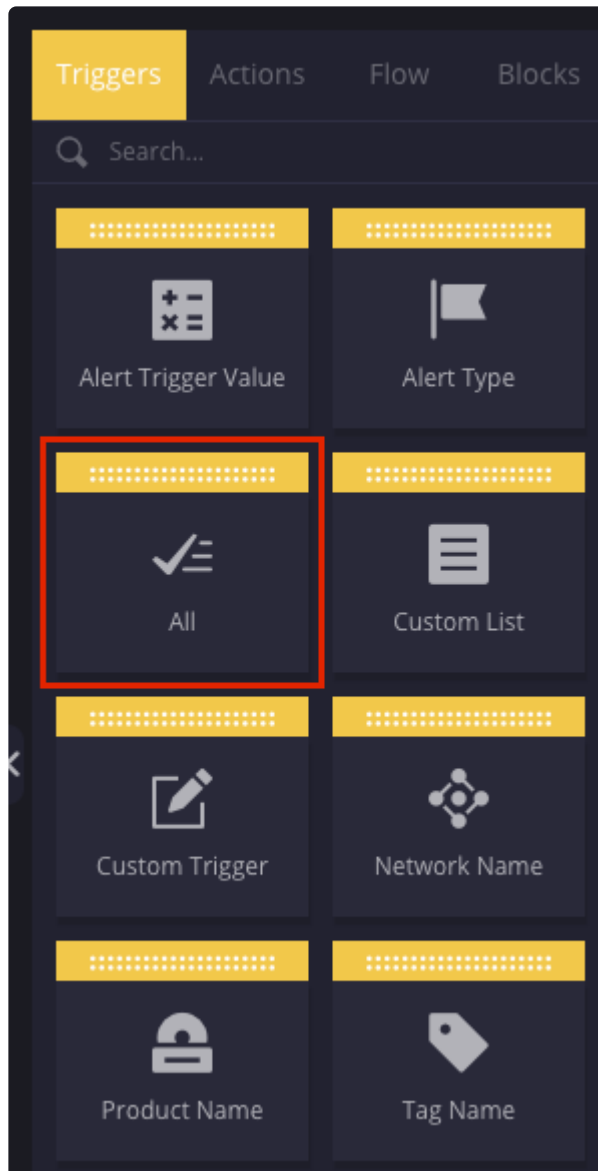



You can also import the premade playbook using the import icon found in the menu icon in the top left part of the playbook designer.

[My First Automation Playbook](#)



6. Each playbook starts with the trigger that will trigger the playbook. Navigate to the Triggers tab and drag the “All” Trigger to the first step of the playbook. The playbook will trigger on every alert ingested into Simplify.



7. We will now begin to form the playbook with the actions we created as part of the “WHOIS XML API” Integration. Navigate to the Actions tab and click on the “WHOIS XML API” drop down. The actions you created will be presented beneath the integration. If the actions are not visible, make sure they are enabled in the IDE module and saved.
8. The First action we will drag into the playbook after the trigger is the “**Get Domain Details**”. Lets customize the action and define the scope we would like the action to run on. As presented in the screenshot below we have chosen to run the action on all the Entities that are URLs and for the Domain name field we have used the placeholder “Entity Identifier”. In order to insert a placeholder click on the placeholder icon  and search for `Entity.Identifier` in the search bar. As mentioned previously, this action will connect to the “WHOIS” site, extract the details of the Domain and present them in a Json format. The parameter we defined for the action `Check Availability` will check if the domain is available or not.

WHOIS XML API - Get Domain Details

WHOIS XML API_Get D...

Description

Manual ☐ Auto ☒

If step fails
Stop playbook ▼

Choose Instance *i* Default Environment_Sys... ▼

Configure Output

Entities *i* All URLs ▼

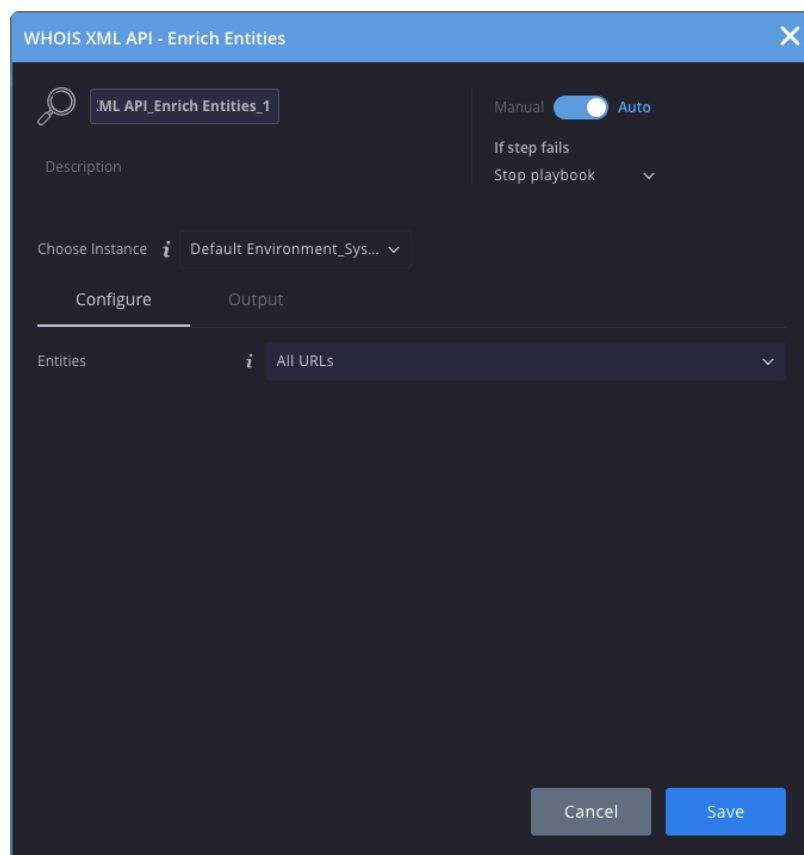
Check availability *i* ☒

[Entity.Identifier] X

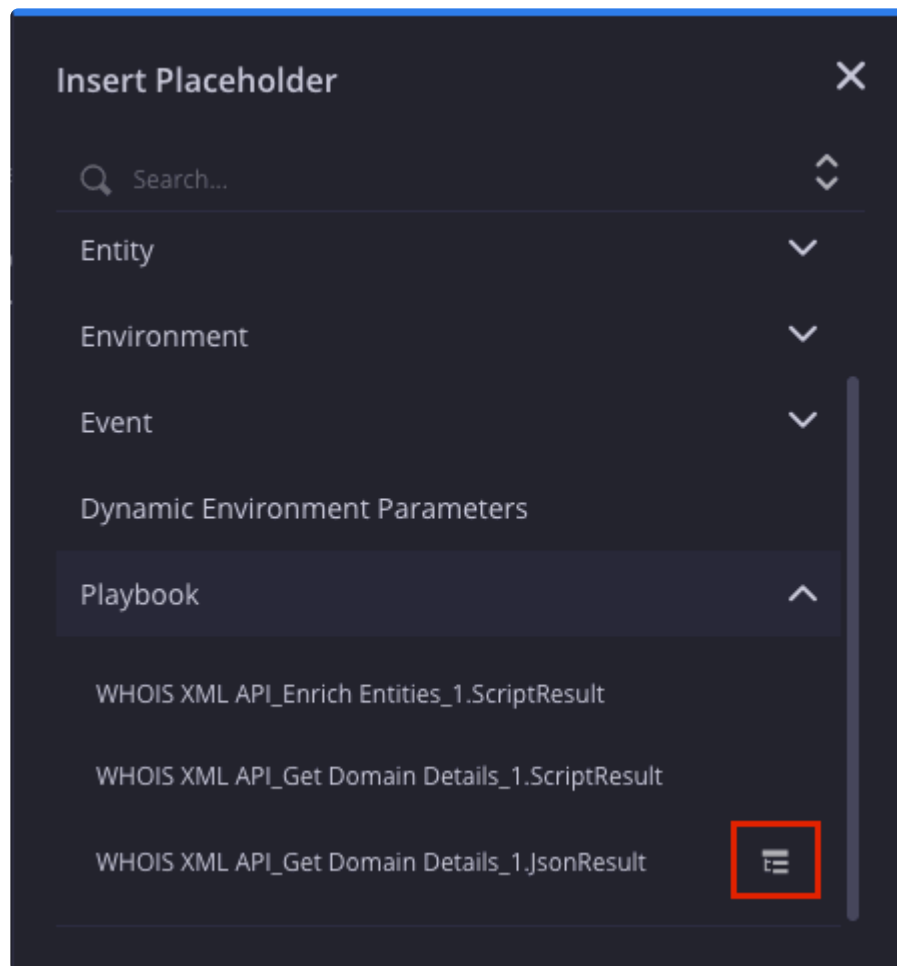
Domain Name * *i*

Cancel Save

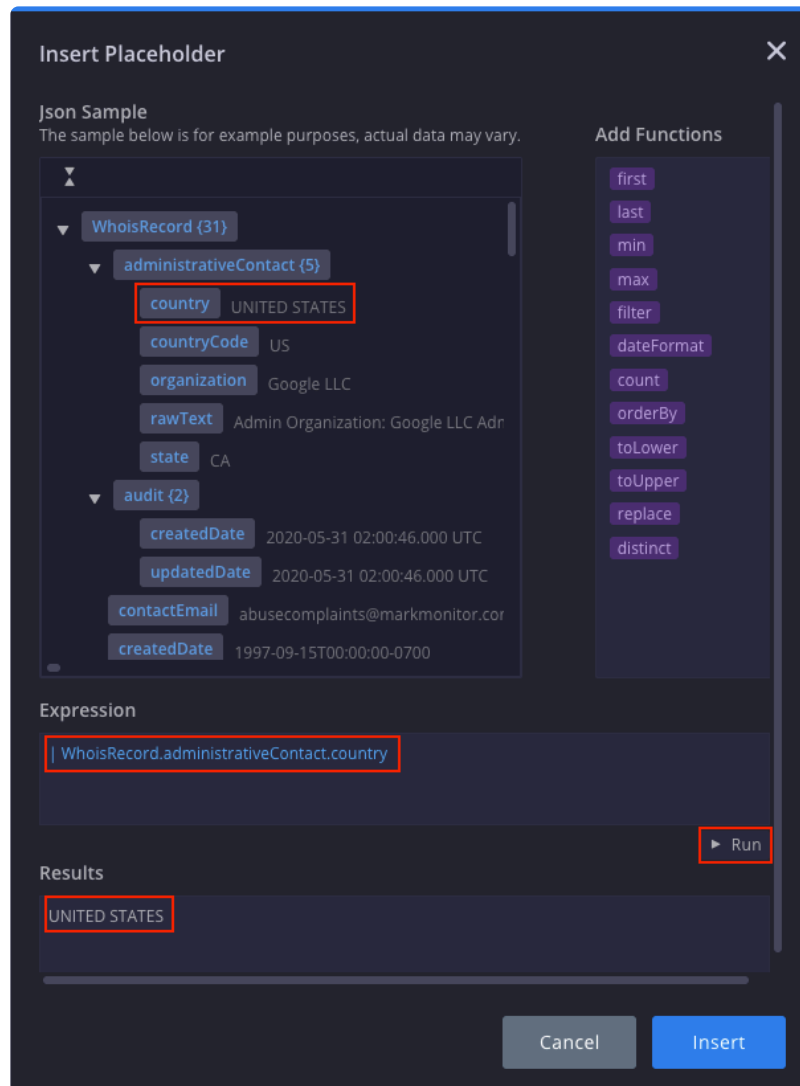
- For the second action in the playbook, drag the “Enrich Entities” action and customize the action to run on “All URLs”. As mentioned in the “My First Action” how to, we created the action to run on a specific entity scope therefor do not need to define the field such as Domain name as done in the previous action.



- For the next action we will use the “Add Entity Insight” action which is part of the Simplify Integration. For the Entity scope we will choose the “All URLs” as done in the previous actions in the playbook and in the Insight itself we will use the **Siemplify Expression builder** to extract the specific field in the Json – Country. In order to open the expression builder, click on the placeholder icon, choose the playbook dropdown and select the icon presented next to WHOIS XML API_Get Domain Details_1.JsonResult



The Json Sample presented in the Expression builder is the Json example we inserted in the IDE as part of the “My First Action” How to. In order to extract the Country field from the Json we will click on the “Country” field in the Json. In order to test the placeholder click on the Run icon and view the result under the “Results” field as shown in the screenshot below.



- Next, we will create an Entity from the country related to the domain in order to run the “Is in custom list” action on that entity in the next step. From the Siemplify Integration drag the “Create Entity” Action into the playbook and configure the action to run on “All URLs” and use the expression builder to insert the country placeholder in the entity identifier field. For the Entity Type choose the Generic Entity type and click on Save.

Siemplify - Create Entity

Siemplify_Create Entit...

Creates an entity and adds to requested alert

Manual ☐ Auto ☒

If step fails
Stop playbook

Choose Instance *i* Default Environment_Sys... ▾

Configure Output

Entities *i* All URLs ▾
[WHOIS XML API_Get Domain Details_1....] []

Entities Identifies * *i*

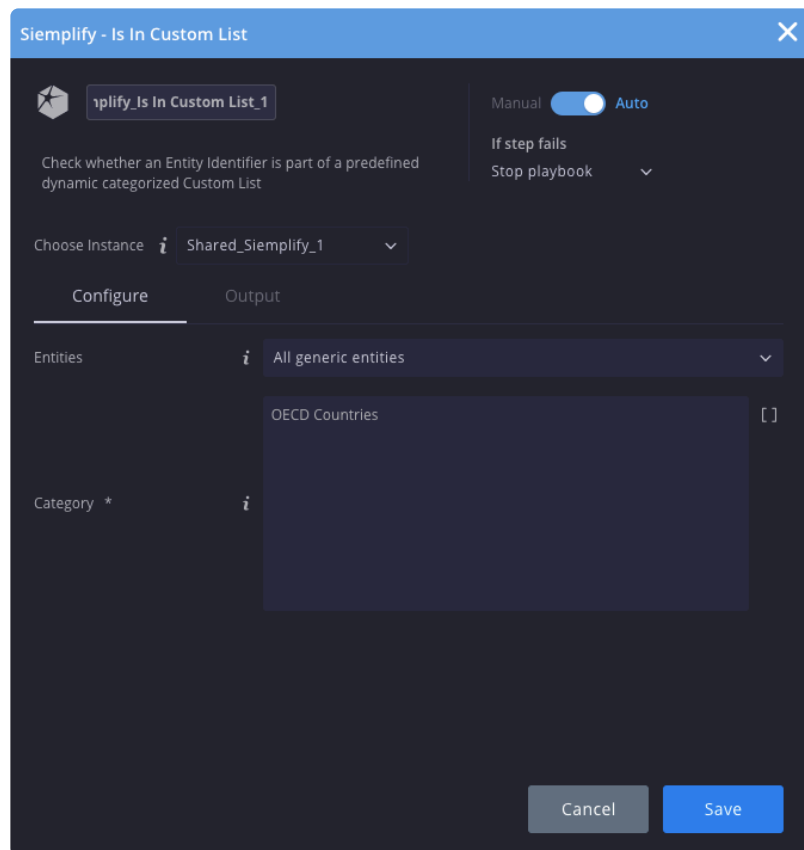
Entity Type * *i* Generic Entity ▾

Is Internal *i* ☐

Is Suspicious *i* ☐

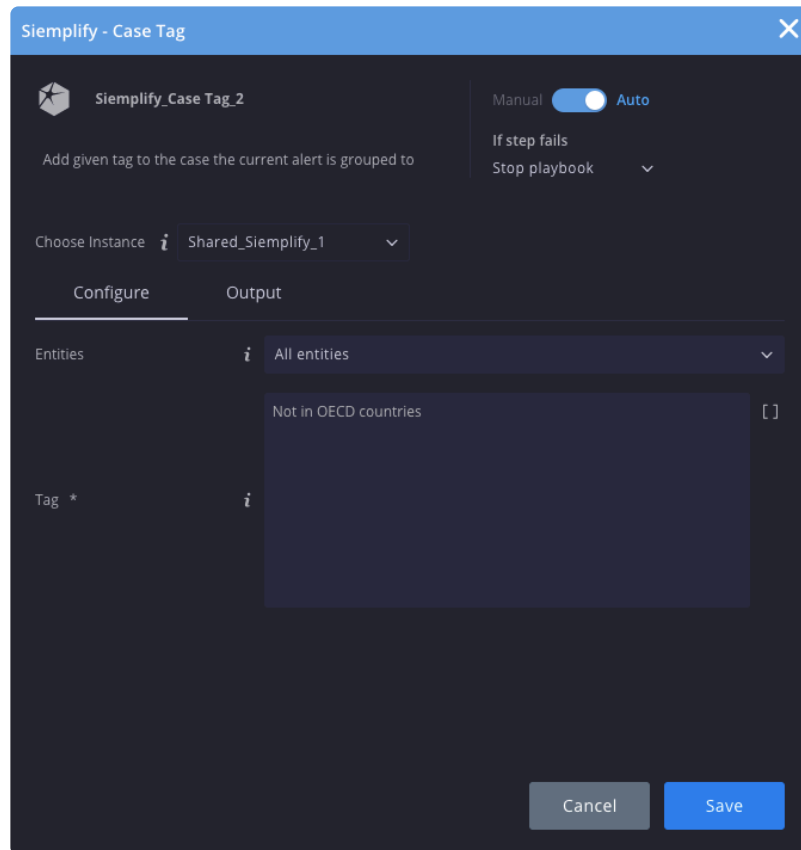
Cancel Save

- For the next action in the playbook we will add the “Is in Custom List” action which will run on all generic entities (the generic entity we created previously) and in the Category we will add the category we configured for the custom list we created as presented in the screenshot below.



- Now we will add the **IF condition** in the playbook to determine if the country related to the Domain name requires the Analyst to further investigate the case. The 1st branch will check if the script result for the "Is in Custom list" returned a false result and the Else branch will go to the opposite result as shown in the screenshot.


14. Once we have added the “IF Condition” action, 2 branches will be created right after it based on the number of branches you defined in the flow action.
15. Lets start by customizing the 1st branch. If the result for the “Check in custom list” returns a false result, it will lead to this branch. If the result is false this means that the country related to the domain is not in the custom list we created for OECD countries, leading us to want to further investigate this case. For the first action in this branch we will drag a “Case Tag” action from the Simplify Integration and add the tag “Not in OECD countries”.



The screenshot shows the 'Siemplify - Case Tag' configuration window. The title bar is blue with a close button. The main area has a dark background. At the top left, there's a star icon and the text 'Siemplify_Case Tag_2'. Below it, a description says 'Add given tag to the case the current alert is grouped to'. To the right, there's a 'Manual' toggle switch (currently off) and an 'Auto' toggle switch (currently on). Below the toggle, there's a section 'If step fails' with a dropdown menu showing 'Stop playbook'. Underneath, there's a 'Choose Instance' dropdown menu with 'Shared_Siemplify_1' selected. Below this, there are two tabs: 'Configure' and 'Output'. The 'Configure' tab is active. It contains a section 'Entities' with a dropdown menu showing 'All entities'. Below this, there's a text input field with the value 'Not in OECD countries'. At the bottom left, there's a 'Tag *' label. At the bottom right, there are 'Cancel' and 'Save' buttons.

16. The next action will be to assign the case to a higher Tier to further investigate this case. In order to do this we will drag the “Assign Case” action to the playbook and choose @Tier2 as the Assigned User.

Siemplify - Assign Case

 Siemplify_Assign Case_1

Manual ☒ Auto

If step fails
Stop playbook ▼

Assign case to specific user or usergroup

Choose Instance ⓘ Shared_Siemplify_1 ▼

Configure

Output

Entities ⓘ All entities ▼


Assigned User * ⓘ @Tier2 ▼

Cancel

Save

17. The last action of this branch will be “Siemplify Change Priority” action in order to change the priority to “High” as shown in the screenshot.

Siemplify - Change Priority

 mplify_Change Priority_1

Manual ☒ Auto

If step fails
Stop playbook ▼

Automatically change case priority to the given input

Choose Instance ⓘ Shared_Siemplify_1 ▼

Configure

Output

Entities ⓘ All entities ▼

Priority * ⓘ High ▼

Cancel

Save

18. Once we have finished with the top branch we will customize the Else branch. As this branch indicates that the country of the domain is in the OECD countries we have decided that it will not require any further investigation. We will first add a tag as done in the 1st branch with a tag “In OECD countries”.

We will then add an additional action that will close the case. All the actions that we have added until now into the playbook have been configured on Automatic mode. As closing a case is a sensitive action we have configured this action to run manually and will require the response of the analyst to execute the action. Add the Reason, root cause and comment in the close case action and save the playbook.

Siemplify - Close Case

Siemplify_Close Case_1

Closes the case the current alert has been grouped to

Manual ☒ Auto

If step fails
Stop playbook

Choose Instance *i* Shared_Siemplify_1

Configure Output

Entities *i* All entities

Reason * *i* NotMalicious

Root Cause * *i* Other

Comment * *i* Was found in OECD countries

Cancel Save

You have now finished customizing your First Automation. In order to see the execution of the playbook navigate to the cases screen, simulate the “Phishing Email” Case and follow the playbook running on the alert and the result of each action in the playbook.

1.7. Publish Your First Integration

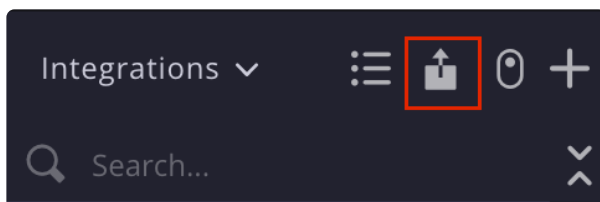
Overview

Once you have developed your first integration you can publish it to the Siemplify Integration Marketplace. This is a great way for users to share their custom integrations with other users. In this “How to” we will go through the steps of how to publish your own custom integration.



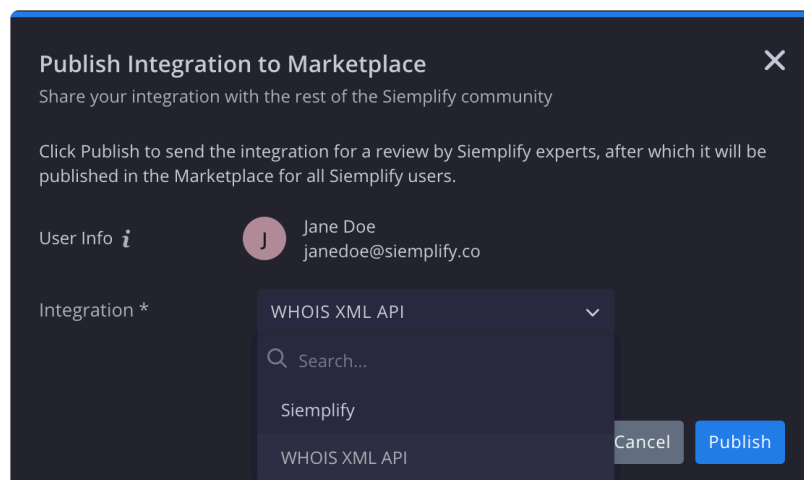
<https://www.youtube.com/embed/IfJcnWNnujs?rel=0>

1. Navigate to the IDE and click on the “Publish integration to Marketplace” icon.

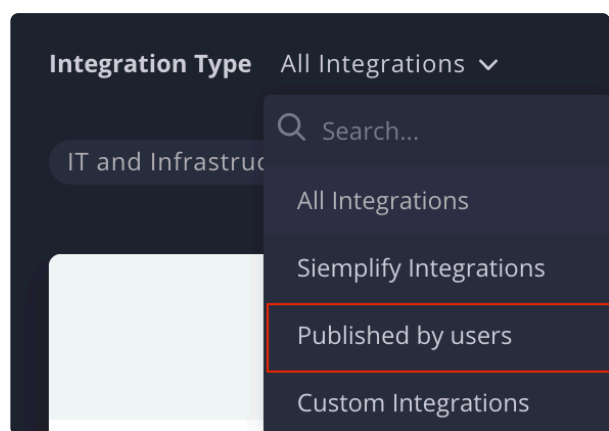


2. Choose the integration you would like to publish from the dropdown and click on the Publish button. The user info presented in the screenshot will be published together with the integration in the Marketplace. Once you click on publish integration, it will be sent for a review by Siemplify experts,

after which it will be published in the Marketplace for all Simplify users.



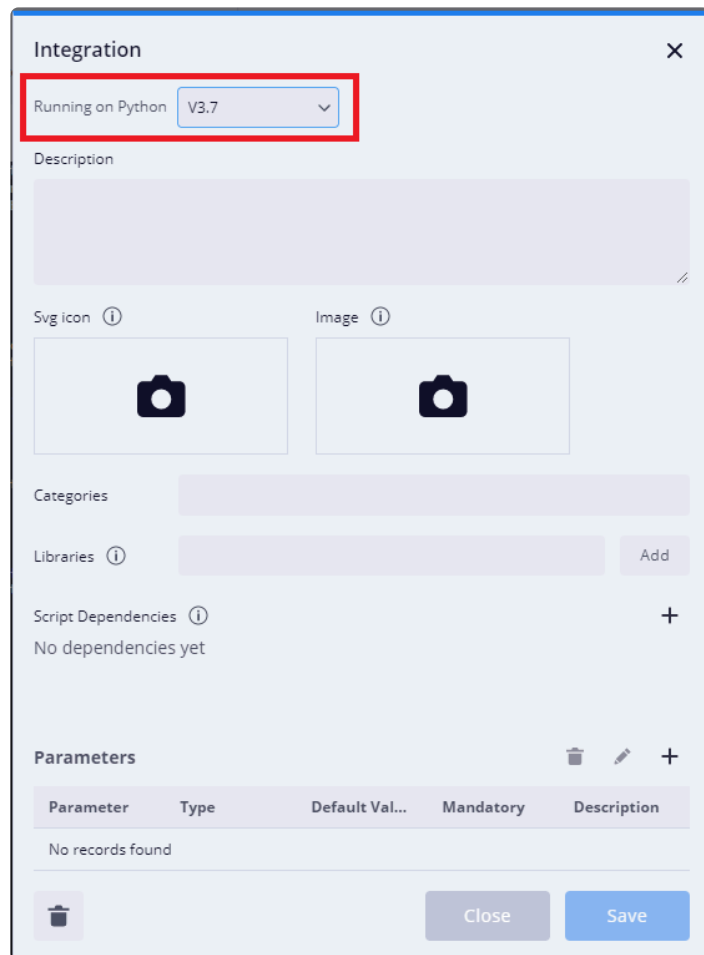
3. Once the integration is published in the Marketplace it can be filtered using the “Filtered by users” filter which will filter all the integrations that were published by the Simplify Community.



1.8. Requirements for Publishing Integration

Integration Requirements:

- **Python 3.7** – we recommend to develop all integrations in Python 3.7.



The screenshot shows the 'Integration' configuration form. At the top, there is a dropdown menu labeled 'Running on Python' with 'V3.7' selected. Below this is a large text area for 'Description'. Further down are two image upload sections: 'Svg icon' and 'Image', each with a camera icon. Below these are 'Categories' and 'Libraries' sections, both with text input fields and an 'Add' button. The 'Script Dependencies' section shows 'No dependencies yet' and a '+' button. The 'Parameters' section at the bottom has a table with columns: Parameter, Type, Default Val..., Mandatory, and Description. The table is currently empty, showing 'No records found'. At the bottom right are 'Close' and 'Save' buttons.

- **Integration Description** – the integration should include a description of the product you have chosen to integrate with.

Integration

Running on Python V3.7

Description

Svg icon

Image

Categories

Libraries

Add

Script Dependencies

+

No dependencies yet

Parameters

Parameter	Type	Default Val...	Mandatory	Description
No records found				

Close

Save

- Icons-

SVG Icon – each integration should be published with an SVG icon that will affect all the integration icons in the platform.

PNG Icon – each integration should also include a PNG icon that will display as the picture presented in the Integration Marketplace.

Integration

Running on Python V3.7

Description

Svg icon ⓘ

Image ⓘ

Categories

Libraries ⓘ

Add

Script Dependencies ⓘ

+

No dependencies yet

Parameters

Parameter

Type

Default Val...

Mandatory

Description

No records found

Close

Save

- **Integration Category** – we recommend defining the integration category to enable other users to filter the integration in the Marketplace by its category (You can select one of the categories from the list in the Marketplace).

Integration

Running on Python V3.7

Description

Svg icon

Image

Categories

Libraries Add

Script Dependencies No dependencies yet

Parameters

Parameter	Type	Default Val...	Mandatory	Description
No records found				

Close Save

Siemplify Cloud

HOME PAGE DASHBOARDS CASES PLAYBOOKS SEARCH REPORTS COMMAND CENTER

7 10 BETA

Siemplify Marketplace

Integrations

Use Cases Power Ups Analytics

Type All Integrations Status All

IT and Infrastructure Management Data Enrichment Network Security SIEM & Log Management

Latest Releases

AlgoSec (v1.0)

Siemplify Integration

Manage your network security effectively, swiftly, and confidently. No matter where your network lives. Gain complete visibility, automate changes, and always be compliant.

DUO (v1.0)

By Community (John DePalma)

Cisco's MFA Solution. Duo is engineered to provide a simple, streamlined login experience for every user and application, and as a cloud-based solution, it integrates easily with your existing...

Google Kubernetes Engine (v1.0)

Siemplify Integration

Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multip...

Webhook (v1.0)

By Community (Shir Savion)

Webhook.site lets you easily inspect, test and automate any incoming HTTP request or e-mail.

SiemplifyUtilities (v9.0)

Siemplify Integration

Siemplify Utilities integration package includes various utilities action.

Filter by categories

Search...

IT and Infrastructure (53)

Management (6)

Data Enrichment (33)

Network Security (33)

SIEM & Log Management (16)

- **Dependencies** – if there is a need to use external libraries, add the dependencies in the integration settings.

Integration

Running on Python V3.7

Description

Svg icon

Image

Categories

Libraries

Add

Script Dependencies

+

No dependencies yet

Parameters

Parameter

Type

Default Val...

Mandatory

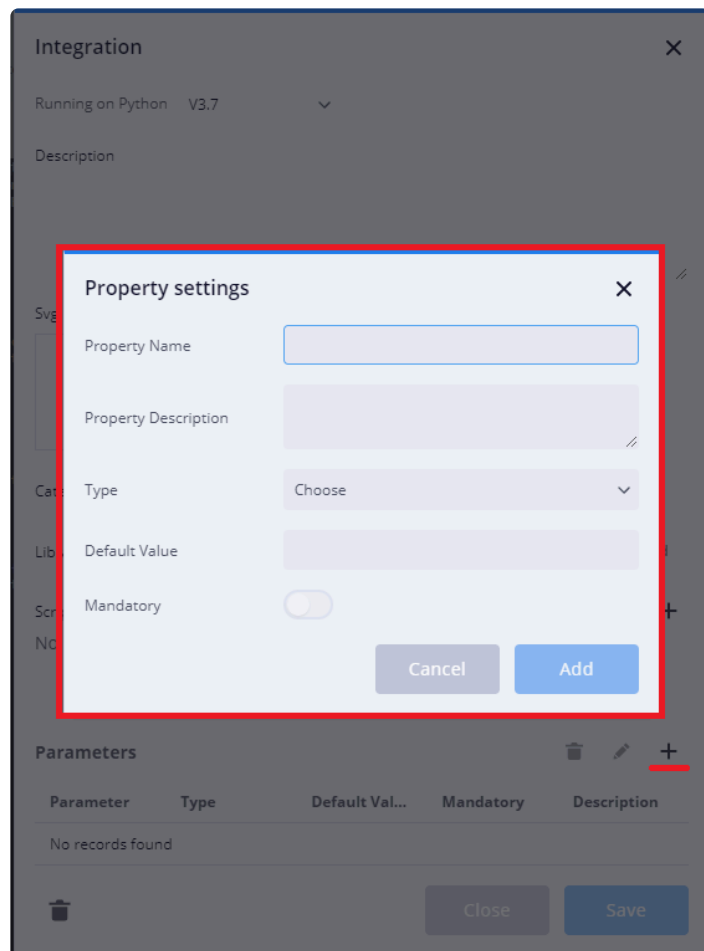
Description

No records found

Close

Save

- **Integration Parameters** – each integration should include the parameters required for a successful integration with the product, including a description of the parameter.



The screenshot shows a 'Property settings' dialog box with the following fields and controls:

- Property Name:** A text input field.
- Property Description:** A text area.
- Type:** A dropdown menu with 'Choose' selected.
- Default Value:** A text input field.
- Mandatory:** A toggle switch.
- Buttons:** 'Cancel' and 'Add' buttons at the bottom right.

The dialog box is titled 'Property settings' and has a close button (X) in the top right corner. The background shows a blurred view of the 'Integration' settings page, including a 'Parameters' table with columns: Parameter, Type, Default Val..., Mandatory, and Description. The table currently shows 'No records found'.

- **Manager** – in order to avoid reusing code it is recommended to add a manager to the integration. A manager is a Python file that can be referenced from every other script in the integration.
- **Ping action** – a ping action is a required action to test a successful connection to the product. The result value of the action should be true when the connection is successful. This action should be disabled since it's not an action that is used in a Playbook.
- **Linux** – the integration should support Centos OS 7 and above.

Action Requirements

- **Action description** – each action should include a description that explains the functionality of the action.

The screenshot shows the Simplify IDE interface. At the top, there's a header with 'Action' and 'Test Integration' tabs. Below the header, a 'Description' field is highlighted with a red box. The main area is split into two panes. The left pane is a code editor showing a Python script for a 'SiemplifyAction'. The script includes imports for 'SiemplifyAction', 'SiemplifyUtils', and 'ScriptResult', and defines a 'main' function that interacts with the Simplify system. The right pane is a configuration panel with tabs for 'Details', 'Testing', and 'Debug Output'. Under 'Details', there are fields for 'Output Name' (set to 'ScriptResult'), 'Include JSON Result' (a toggle switch), 'Timeout configuration' (with fields for Days, Hours, Minutes, and Seconds), 'Default Return Value', and a 'Parameters' table. The 'Parameters' table is currently empty, showing 'No records found'.

- **Action structure** – it is recommended to follow the template presented in the IDE when creating a new action.
- **Action parameters** – each action should include the parameters relevant to the action, including a description explaining the parameter. Make sure you match the type of the parameter according to the requirements of the action.

The screenshot shows a 'Parameter Settings' dialog box. It has a close button (X) in the top right corner. The form contains the following fields:

- Parameter Name ***: A text input field with a red error message 'This field is required' below it.
- Description ***: A text area with a red error message 'This field is required' below it.
- Mandatory**: A toggle switch that is currently turned on (green).
- Type ***: A dropdown menu with 'Choose' selected.
- Default Value**: A text input field.

 At the bottom of the dialog are 'Cancel' and 'Save' buttons.






- **Running action on a context of an alert** – it is recommended to create the actions in the context of an alert. This means applying the logic so that the action will allow running on a specific scope of entities, for example on URL entities. This can be done by using the `siemplify.target_entities`

method which returns a list of all the target entities in the scope we have chosen to run the action on. An example of implementation can be found in the article [“My first Action”](#).

The screenshot displays the Siemplify IDE interface. On the left, the Python editor (Python V3.7) shows a script for URL scanning. The script imports necessary modules, defines a function to start an operation, and handles the execution flow, including submitting a scan and logging results. On the right, the 'Parameters' panel is visible, showing a dropdown menu for selecting a 'Scope'. The dropdown list includes options like 'Source entities', 'Destination entities', 'Internal users', 'External users', 'All hostnames', 'Internal hosts', 'All URLs', 'External hosts', 'All IP addresses', and 'Internal IP addresses'. A red box highlights the dropdown menu, and a blue 'Help' button is visible on the right side of the panel.

- **JSON Result** – for actions that return data, the action should return a JSON result by using the function [add_result_json](#).
- **Add JSON Example** – it is recommended to add a JSON example that can be used in the expression builder when creating a playbook using your integration. This can be done by clicking on the JSON icon in the IDE and importing your JSON example.

Manage JSON Sample

Details Testing Debug Output

Output Name * ScriptResult

Include JSON Result ☒

Timeout configuration

Script Timeout *


0 Days





0 Hours

5 Minutes



0 Seconds

Default Return Value

Parameters 

Parameter	Type	Default Value	Mandatory	Description
User Type	List	Basic	✓	
Email	String	email@gmail.com	✓	
Last Name	String	LastName	✓	
First Name	String	FirstName	✓	

JSON Result Sample

Import JSON Example  

Import or export JSON sample for the current action

< >

> groups [1]

id k&KdKr6TLWuxdDtk0hjSzL

> meta {3}

> name {2}

> phoneNumbers [1]

> roles [1]

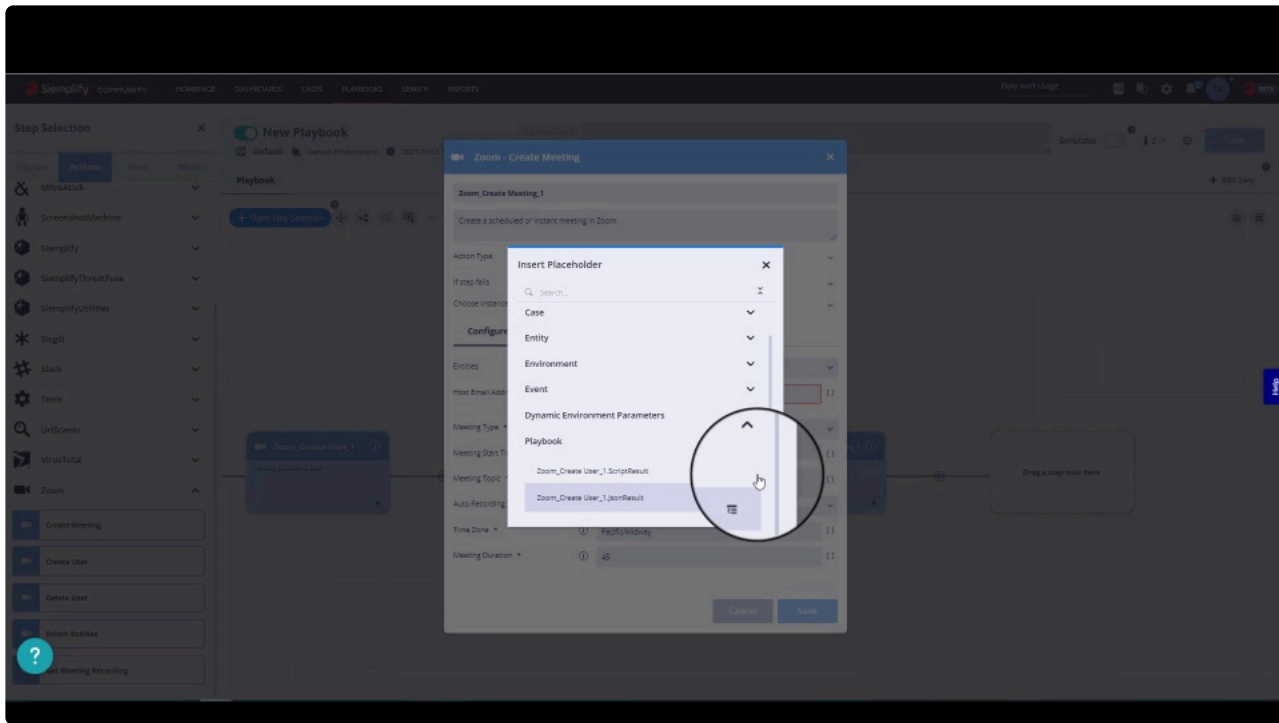
> schemas [1]

> urn:ietf:params:scim:schemas:extension:enterprise:2.0:User {2}

> urn:us:zoom:scim:schemas:extension:1.0:ZoomUser {1}

Done

The JSON example enables the user to use the JSON result values as “placeholders” in a playbook. An example can be seen in the attached video



<https://fast.wistia.net/embed/iframe/80iv0tmitf>

- **Enrich Entities** – if enrichment is relevant to the action we recommend adding an enrichment step in the action to enrich the entities with the data coming from the product you integrated with. It is highly recommended to add a prefix to the enrichment field keys.

For example, suppose we want to enrich an entity with the following data:

```
entity_enrichment = {"first_name": "First Name", "last_name": "Last Name"}
```

First, you have to make sure that the dictionary is not nested and has only one hierarchy in it.

Then, add the product name as a prefix.

For example, in the following code we are adding the prefix “Zoom” to the new fields that were added by the enrichment action

```
entity_enrichment=add_prefix_to_dict(entity_enrichment, "Zoom")
```

Then, update the additional properties of the specific entity by using the method: `entity.additional_properties.update(entity_enrichment)`

Once the entity’s additional properties were updated we will add them to the alert by using this method: `simplify.update_entities(enriched_entities)`

You can see the entity’s full details by clicking on it.

The screenshot displays the Simplify web application interface. The main header includes navigation links: SIMPLIFY, COMMUNITY, HOMEPAGE, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, and REPORTS. A 'Daily Alert List' button is visible in the top right.

The left sidebar shows a list of 14 cases, including 'Phishing email detector' and 'Website Certificate Validation...'. The main content area is titled 'Phishing email detector' and shows details for ID 964, Default Environment, and a timestamp of 2021-10-14 16:02:19. It includes a 'Simulated Case' tab and a 'Manage Tags' button.

The 'Indicators Information (4)' section lists several indicators, including 'VICKIE.B@SIMPLIFY.CO', 'HTTP://MARKOSSOLOMON.COM/YIQ...', 'YOUR NEW SALARY NOTIFICATION', and 'F.ATTACKER4@GMAIL.COM'. The 'F.ATTACKER4@GMAIL.COM' indicator is selected, and its details are shown in a modal window on the right.

The modal window for 'F.ATTACKER4@GMAIL.COM' displays the following metadata:

IsEnriched	True
IsTestCase	False
TFuse_tags	Trickbot
TFuse_type	email
TFuse_subtype	mal_email
IsVulnerable	False
TFuse_report	https://simplify.threatst...
TFuse_status	inactive
TFuse_feed_id	0
TFuse_severity	low
IsInternalAsset	False
IsFromLdapstring	False
TFuse_confidence	0
OriginalIdentifier	f.attacker4@gmail.com
TFuse_threat_score	20
TFuse_creation_time	2017-06-29 19:00:19
TFuse_expiration_time	2017-09-27 18:28:35
TFuse_modification_time	2017-09-28 16:40:12
TFuse_source_confidence	100
Testintegration_last_name	Last Name
Testintegration_first_name	First Name

The modal window also includes a 'Close' button and a 'Help' button.

<https://fast.wistia.net/embed/iframe/h2wbhykffa>

- **Logging** – it is highly important to add logs, especially in complicated actions. Every exception or error should be logged with the appropriate level. (e.g. [info](#), [warn](#), [error exception](#))

1.9. My First Use Case

Overview

What is a Use Case?

A Use Case is a package of items that together provide a solution (e.g. automating phishing threats, reducing false positives, orchestrating incident investigations, etc.).

Once a Use Case is published to the Simplify marketplace it is available for all Simplify users to use.

A Use Case package consists of Test Cases, Connectors, Playbooks, and also Integrations and rules of mapping & modeling.

1.9.1. Creating a Use Case

Step by Step Guide



1. Define the Use Case

Write a description of the security threat you are solving with the use case. Define what kind of alert will be handled and what is the detection product that generates it.

For example, CrowdStrike – Falcon Overwatch via Malicious Activity.

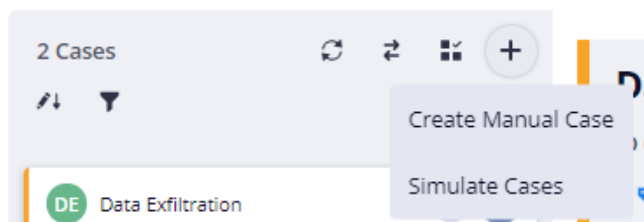
The next thing you should do is draw an incident response, orchestration, or automation process, to handle this alert.

2. Prepare Use Case Alerts

You can create a custom Alert / Event according to a real data case.

Generate sample security alerts/events from a detection tool to simulate the use case.

Go to Cases, and under the plus sign click on “Simulate Cases”.



Then, click on the plus sign in the opened window.



Why Create a Simulation Alert?

When you create a Simulation Alert, you can always use it to test the playbook and the use case. Also, this simulation will be part of the use case package.

How do you simulate an alert?

Fill in the fields of the simulation alert based on the alerts you prepared for the use case.

Simulation Alert Fields:

 **Add New Case** 

Source \ SIEM Name

Arcsight

Rule Name

Data Exfiltration

Alert Product

DLP_Product

Alert Name

Data Exfiltration

Event Name

Data Exfiltration

Additional Alert Fields


+

Additional Event Fields

+


applicationProtocol

TCP




destinationAddress

10.0.0.28




detinationHostName

lab@Siemplify.local



destinationUserName

XWzNr1l@gmail.com



Cancel

Save

Next, you need to create a simulation alert in Siemplify, based on your sample alert/event.

- **“Source \ SIEM Name”:**

Displays the source of the alert, be it a SIEM or another detection tool.

Example: This field has the value “Arcsight”, a SIEM product.

If the alerts are generated by the product itself, and Siemplify pulls it from there- add the product name here.

- **“Rule Name”:**

Displays the SIEM rule that generated the alert.

Example – This field has the value “Data Exfiltration” which is a SIEM rule.

If no SIEM is involved, just add the name of the alert generated by the detection product.

- **“Alert Product”:**

Displays the detection tool that generated the alert.

Example – The Alert Product is a DLP (Data Loss Prevention) product.

- **“Alert Name”:**

Displays the name of the alert as generated by the product.

Example – The alert name is “Data Exfiltration”. (Meaning, unauthorized movement of data of any sort).

- **“Event Name”:**

Displays the name of the base event that triggered the alert.

Example – The event name is “Data Exfiltration” since it is also the name of the event.

- **“Additional Alert Fields”:**

Displays usually an alert is generated by a SIEM, It displays additional content for easier incident response.

Example 1 – SIEM fields like Severity, Impact, Sensitive Assets, etc.

Example 2 – If no SIEM is involved, just add one field with the name of the alert (alert_name:).

- **“Additional Event Fields”:**

Displays all the raw security data used in incident response. Add here all the data from the sample alert you are using for the use case.

Use the exact schema of fields found in the sample alert.

Most Common Use – Put here the security data from your alert (e.g src_ip, dest_port, email_headers, etc.)

3. Extract Entities (Map & Model the data)

Select the visualization model of the alert (the entities Simplify should extract and the relations between them), and map the raw data fields into the selected model.

Entity Fields

Rule Level	Target field	Extracted Field	Alternative Field 1	Alternative Field 2	Extraction Func	Extraction Param	Transformation Fu...	Transformation Pa...
Source	<div><div></div><div>DestinationHos...</div></div>	destinationHostName	destination_hostName		Delimiter	,	TO_STRING	
Source	<div><div></div><div>USB</div></div>	USB			Delimiter	,	TO_STRING	
Source	<div><div></div><div>DestinationAdd...</div></div>	destinationAddress	destination_address		Delimiter	,	TO_IP_ADDRESS	
Source	<div><div></div><div>DestinationUse...</div></div>	destinationUserName	destination_userName		Delimiter	,	TO_STRING	

Help

You can get here by clicking on the configuration icon on the event (As seen in the below screenshot). More information on how-to can be found here- [Getting Started with Simplify](#), [Create Entities](#), [Mapping & Modeling](#).

Data Exfiltration

ID 653 | Default Environment | 2021-09-20 11:22:09 | Triage

Manage Tags

11 @Tier1

Explore

1. DATA EXFILTRATION (1) • 2021-09-20 11:22:08

Overview | **Events (1)** | Playbooks (0)

Name	Type	Source / Product	Artifacts	Port	Outcome	Time	Configure Event
Data Exfiltration	Data Exfiltration	DLP_Product				2021-09-20 11:22:08	View More

The next thing you should do is check if all the entities are created accordingly.

Entities Highlights (3) ⓘ

LAB@SIEMPLIFY.LOCAL View more ⚙️

10.0.0.8 View more ⚙️

XWZNR1L@GMAIL.COM View more ⚙️

You can watch the entities under the case tab, “Entities Highlights” (As seen in the screenshot below). Click “View More” on each entity to make sure the mapping is properly configured.


4. Build a Playbook

First, you want to define the incident response flow for the alert, be it a chart or a drawing. Then, design the flow you defined as a Siemplify playbook. To do so, you need to download and configure the integrations you would like to use in the playbook. See here: [Siemplify Integration Marketplace](#), [Configure Integrations](#)

h4. Configuring Actions in the Playbook

- “Action Type” – Select whether this action should run automatically or manually (wait for a human approval)
- “Choose Instance” – Select Dynamic
- “If Step Fails” – Choose whether the playbook will stop if the action fails or it will skip to the next action.
- “Entities” – Select what type of entities this action should affect (of those you extracted in your simulation alert).

Other parameters – Fill in the action-specific parameters based on the documentation of the integration

 **Simplify - Case Tag** ×

Suspicious Tag_1

Add given tag to the case the current alert is grouped to

Action Type

Automatic

▼

If step fails

Stop playbook

▼

Choose Instance *

i

 Shared_Simplify_1

▼

Configure

Output

Entities

i

 All entities

▼

Tag *

i

 Suspicious

[]

Cancel

Save

Configuring Conditions in the Playbook

Determine the amount of branches – add branches with the “Add Branch” button.

For each branch define the conditions that will trigger this branch.

Use placeholders (square brackets) to reference conditions to Event data, Previous Action results, and more.

Condition

Is Certificate Expired?

This condition determines the progress of the playbook. Conditions are built based on cases data (cases, alerts, vents, entities and environment properties) as-well as data that comes back from previous playbook steps.

Entities

All entities

Parameters

+ Add Branch

1 Cert_Expired

This branch was selected 18 out of 90 runs

Logical Operator And

< 1

[Get Certificate Details.JsonResult| "days_to_expiration"]

+ Branch "Else"

This branch was selected 72 out of 90 runs

Cancel

Save

Important note – Use tools you can actually test in your flow.

Test on live data – Set up a connector that can pull alerts similar to the example alert you created for simulation. [Configuring the Connector](#).

To Test The Connector:

1. First Save the configuration of the connector.
2. Click on **“Run Connector Once”** to make it pull an alert from the source.

The screenshot shows the Simplify web interface. On the left sidebar, under 'Connectors', the 'Website Certificate Validation' connector is selected. The main panel shows the configuration for this connector, with the 'Testing' tab active. A red box highlights the 'Run connector once' button. Below this, there is a table for 'Sample Alerts' with columns: Alert Name, Product, Start Time, End Time, Event Count, and Preview. The 'Output' section is also visible on the right.

3. “Sample Alerts” will show an alert you can ingest into Siemplify.

The screenshot shows the Siemplify interface with the 'Website Certificate Validation' connector selected. The 'Testing' tab is active, displaying a table of sample alerts. The table has columns: Alert Name, Product, Start Time, End Time, Event Count, and Preview. There are five rows of sample alerts, each with a checkbox and a 'Load to system' button. The 'Output' section on the right shows the results of the test, indicating that the connector ingests records from a given table in AirTable.

Alert Name	Product	Start Time	End Time	Event Count	Preview
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	

4. “Output” will show the script logs to indicate the success or failure of the execution.

The screenshot shows the Siemplify interface with the 'Website Certificate Validation' connector selected. The 'Testing' tab is active, displaying the 'Output' section. The 'Output' section shows the results of the test, indicating that the connector ingests records from a given table in AirTable. The 'Sample Alerts' table is also visible on the left.

Alert Name	Product	Start Time	End Time	Event Count	Preview
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	
Website Certi...	AirTable	2021-10-17 13:4	2021-10-17 13:4	1	

More information regarding testing the connector can be found [here](#), with an example of an Email connector with a Phishing Email alert.

Be sure to verify that the same mapping applies to the real alert so that Siemplify is able to extract the relevant entities. Also, make sure that the playbook runs end to end on the alert and performs the defined logic. (try both with malicious and non-malicious alerts).

5. Write a guide

The Use Case you’re creating will be used by other Siemplify users. In order to improve their experience, it’s highly recommended to attach additional content to each use case, in which you should:

- Explain the use case and its value to the SOC.
- Provide recommendations to further improve the Use Case.
- Explain in a few words how to run the use case with simulation data.
- Guide the user about how to run the use case on actual data generated by them.
- Explain How to get free licenses for the tools in use (if there are such).
- Include a How-to on setting up the connector.

The guide can be attached in the “Publish Use Case”, later on.

6. Publish Use Case

It's time to assemble your Use Case- Go to the Use Cases Marketplace, and choose “Create New Use Case” under the hamburger icon on the bottom right.

The screenshot displays the Siemplify Marketplace interface. At the top, the 'Use Cases' tab is selected among 'Integrations', 'Power Ups', and 'Analytics'. The main content area features a card for 'Check Point - Malware Analysis and Response'. The card includes a red warning triangle icon with a bug, a description stating it combines three Check Point products for IOC analysis, and a blue 'Run Use Case' button. To the right of the card is a video player showing a Siemplify and Check Point logo. Below the card, a horizontal navigation bar contains tags for 'Logon', 'Cloud', 'EDR', 'Process', and 'Malware'. A search bar and a hamburger menu icon are located at the bottom right of the interface.


Create Use Case

Create New Use Case

Name *

Category *

Description *


B **U**  ***I***

Powered by Tiny

Video link

Elements

+



Cancel

Save

In the opened window, fill in the details and add the items you developed – Test cases, Playbooks, and Connectors.

In the description category, you can add the guide you've previously written. If it is too long, you can write a short description and attach a link to your full guide.

Publish Use Case

×

Share your use case with the rest of the Siemplify community

Click Publish to send the Use Case for a review by Siemplify experts, after which it will be published in the Marketplace for all Siemplify users.

User Details i

Image

SA

Name *

SHIR.SAVION@SIEMPLIFY.CO Admin

Email *

SHIR.SAVION@SIEMPLIFY.CO@domain.com

Use Case

Select Use Case *

Choose ▾

Add Information *

Description

Cancel

Publish

Now, before you click save- you can export the Use Case. And after that, you can click save. But, don't worry – You can also export it later.

So after you click Save, you can export the package as a ZIP file, import it for testing, And finally, if all goes well-Publish the Use Case to submit it for approval.

CONGRATULATIONS ON YOUR FIRST USE CASE!

*

1.9.2. Requirements for Publishing Use Case

- The simulation alerts in the use case are based on real alerts from a real product.
- All entities are extracted when running the simulation alert in a clean environment.
- All entities are extracted when running the real alert with the connector.
- The playbook runs end to end without errors.
- The final delivery is a ZIP, export that can be imported without errors into the use case marketplace.
- When deployed, all user has to do is configured the integrations to make the playbook run end to end with simulation alerts.

2. Playbook Lifecycle Management

The Siemplify user-friendly playbook builder was designed to enable anyone on your security team to build powerful playbooks with ease. Instead of requiring coding expertise, users select predefined actions from 200+ supported security, IT and third-party tools and snap them into place on the playbook grid.

From the variety of trigger and branching options to built-in data parsing, playbook nesting and more, users find the simplicity of the Siemplify playbook designer a true game changer.

As your SOAR implementation matures and increases in value, your playbook library will develop and grow as well. Siemplify enables a full playbook lifecycle management process which makes maintaining, optimizing and troubleshooting playbooks at scale simple and easy.

Unique capabilities such as playbook run analytics, reusable playbook “blocks”, playbook versioning and rollback and the Playbook Simulator feature ensure your SOAR implementation grows in value, not complexity.

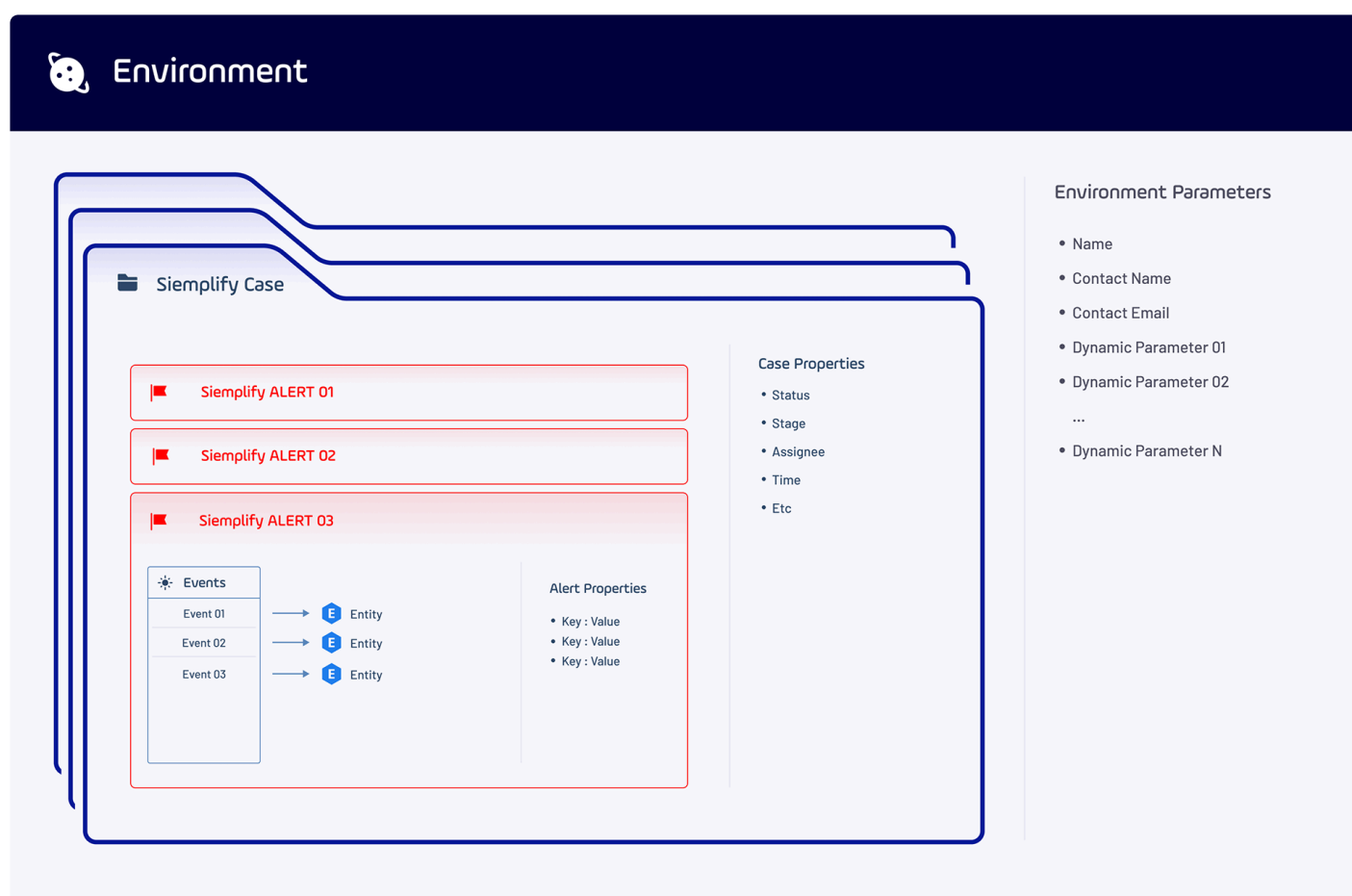
This guide focuses on the design and implementation of automated playbooks using blocks for scale and management alongside the Playbook Simulator for testing and training.

2.1. Prerequisites

For the purposes of understanding the Best Practices playbook guide, it is important to familiarize yourself with the basic Simplify data model. Please make sure to know and understand the following objects and their relations:

- Case
- Alert
- Event
- Entity

The following diagram provides a visual representation of the data model:



2.2. Basic Playbook Design

This section is dedicated to the ‘most common’ playbook design. We will guide you through preparation, features and tools to the point where you feel comfortable enough to implement a playbook using out-of-the-box capabilities. For more advanced customization of the system, please refer to the “Advanced” section.

Preparation

While the data model is the core, the playbook’s role is to shape and edit the data, so before you dive into the process of designing and developing a playbook, you should know and understand the data you are working with.

It is recommended to focus on a specific type of alert at a time, construct a working flow for that alert, and expand from there. Though working on a single alert type, please keep in mind that Siemplify supports blocks (nested playbooks) in your flow. Designing your automated flow with that in mind will help later on with the expansion to other alert types.

When we refer to “Alert Type” in this guide, we mean a ‘family’ of alerts. For example, “User A multiple failed logins” is a specific alert in the “Multiple failed login” family. Your playbook should aim to automate any “Multiple failed logins” alert, regardless of the specific user that failed to login in your example alert.

2.2.1. Know your Alerts

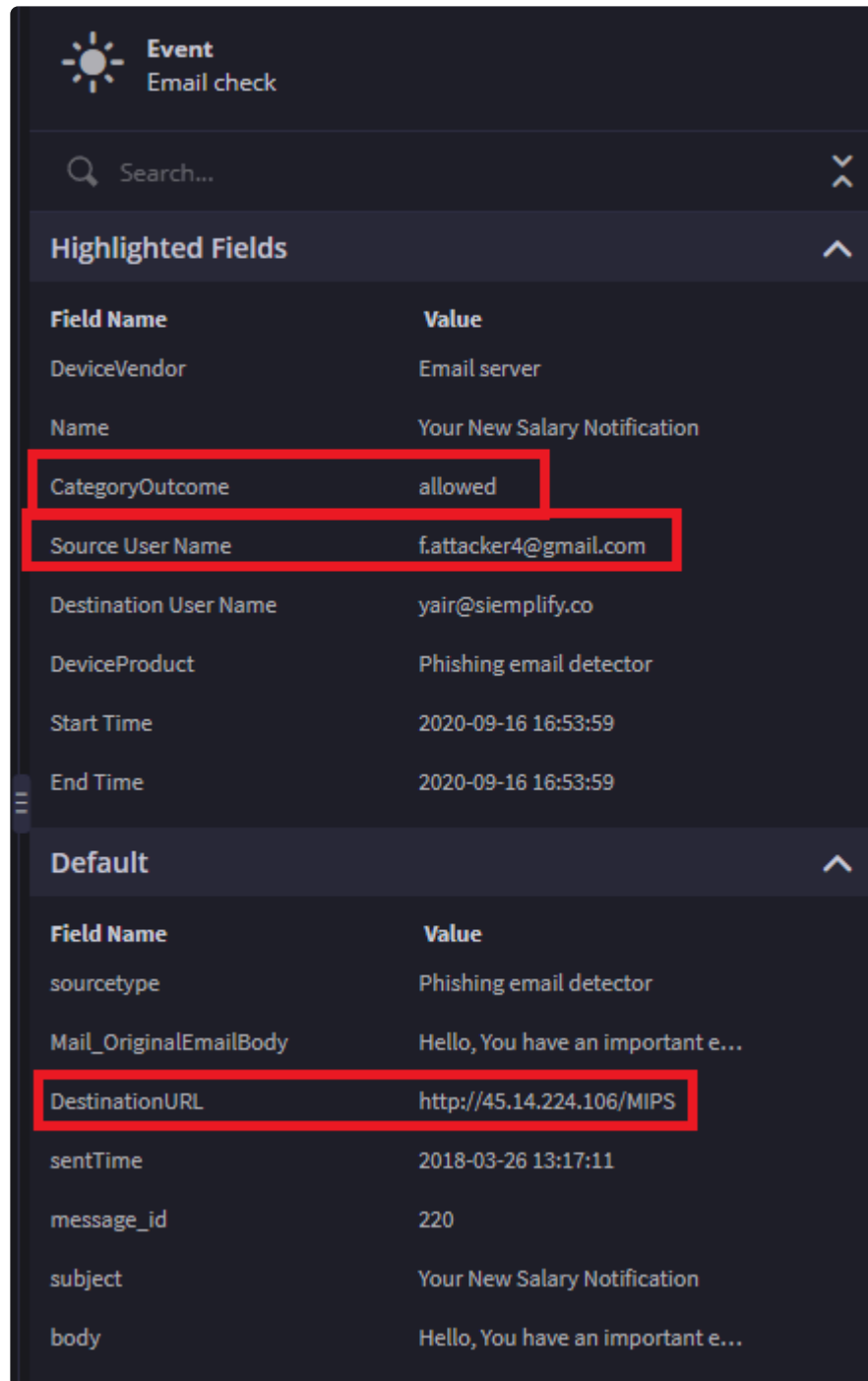
The first step you should take is to choose which alert-type you are going to automate now. If we continue with the example above, let's say we work with "Multiple failed login" alert-type. Once this is established, you should gather at least 2-3 samples of this alert type and note down the relevant cases. You should try and categorize the different properties of the alerts you've gathered. Look for similarities (common properties) and for differences (what makes one alert differ from the rest of that group). When you start building your playbook you are most likely going to use those different properties to design your flow (logic) or simply to act on the proper entities.

Please note that those differences in properties are mostly used for decisions, while actions usually come in sequences. For example, let's say you are going to run a sequence of enrichment actions, then take a decision, then run a sequence or remediation actions. These two sequences can and should be created as blocks (to be reused for other types of alerts in different combinations), and the various alert based decisions of which sequence to execute usually come as part of the playbook itself.

Ideally, you should gather as many different alerts of that type as possible, to make sure your designed flow has no flaws.

Once you have listed these alerts, make sure to explore and understand the scenario, and decide which events are involved (and how many) and what are the main components that are either relevant for decisions or for the flow itself. Usually, IP addresses, hostname/domains, file hashes, URLs, 'types', 'actions' and so on. Don't worry if you are not sure what's relevant, later during the planning phase and the development phase you will have time (and it is recommended) to go back to the alerts 'raw-data' and observe what other pieces of information might be used.

Here's an example of 'relevant' data from a "Phishing" example ("Phishing would be the alert-type and this is an instance of that type as an example):



The screenshot displays the 'Event Email check' interface in Siemplify. It features a search bar at the top, followed by two expandable sections: 'Highlighted Fields' and 'Default'. Each section contains a table of key-value pairs. In the 'Highlighted Fields' section, the rows for 'CategoryOutcome' (allowed), 'Source User Name' (f.attacker4@gmail.com), and 'Destination User Name' (yair@siemplify.co) are highlighted with red boxes. In the 'Default' section, the row for 'DestinationURL' (http://45.14.224.106/MIPS) is highlighted with a red box.

Field Name	Value
DeviceVendor	Email server
Name	Your New Salary Notification
CategoryOutcome	allowed
Source User Name	f.attacker4@gmail.com
Destination User Name	yair@siemplify.co
DeviceProduct	Phishing email detector
Start Time	2020-09-16 16:53:59
End Time	2020-09-16 16:53:59

Field Name	Value
sourcetype	Phishing email detector
Mail_OriginalEmailBody	Hello, You have an important e...
DestinationURL	http://45.14.224.106/MIPS
sentTime	2018-03-26 13:17:11
message_id	220
subject	Your New Salary Notification
body	Hello, You have an important e...

In the image above we see the context details of an event (taken from a “Phishing” alert). Here we see all key-value pairs available in Siemplify for that event. Usually, security events will have all sorts of different pieces of information, most of which are irrelevant. In the planning phase you should attempt and locate relevant fields from the event(s) that will help you take proper decisions and actions automatically.

2.2.2. Analyze existing manual flow

After understanding the alert, it is time to understand the process that analysts take to deal with it.

Collaborate with Analysts – It is recommended that every attempt at constructing an automated process starts by consulting the professional personnel to understand how they deal with these alerts today (either manually or with a different automation system) and what would be their desirable/ultimate process with Siemplify. Usually, these are not the same. Unfortunately, human users greatly differ from automatic processes, and you might have to translate their process into something more manageable. For example, users tend to take actions as needed, and not always in the same order. This phenomenon might contradict the idea of using blocks (reuse logic), because it will seem that each alert, or even each user does something different, where in fact they do practically the same thing. It is your task to find the lowest common denominator and design the automation in such a way that will allow you to both utilize blocks in your flows as well as satisfy end user needs.

Try to understand the flow yourself and construct (with their help) a chart that describes the new desired flow. You can use a workflow tool like Visio to construct that chart. This is also recommended as a documentation process for the playbook, for other users (and yourself) in the future to better understand the flow. Some elements are going to change and it makes the transition that much easier if everyone knows what to expect.

Siemplify recommends constructing a list of “must-do” actions that are going to make their way into your automation. For example, if the process includes sending notifications to customers or creating new tickets on a ticketing system, these would be core actions because you **MUST** notify the customer, or the only way to ‘act’ is by creating a ticket to the appropriate team.

It is important to remember that when you automate, you can ‘afford’ to do a little extra than what the analysts are doing, but you should still keep track of what are the essential demands from the analysts and make sure you give them their proper place.

This is also the place to make sure the expectations of all involved parties are clearly presented. You should make sure that the analysts you design the automation for understand the end result of your work, that they understand how the new process will look and what their role is in it. Once all teams are on the same page, you can continue to the next steps.

2.2.3. Begin Playbook Design

Now it's time to 'translate' the complete flow into Simplify's language. You should have a clear understanding of the flow and the data you have to work with at this point.

Frame required steps

Go over each step of the flow and identify the action(s) in Simplify needed to achieve it. If this is your first time dealing with these actions, it is highly recommended you play around with these actions in a test case/ environment to be familiar with the way they work and their outputs.

You should attempt to understand:

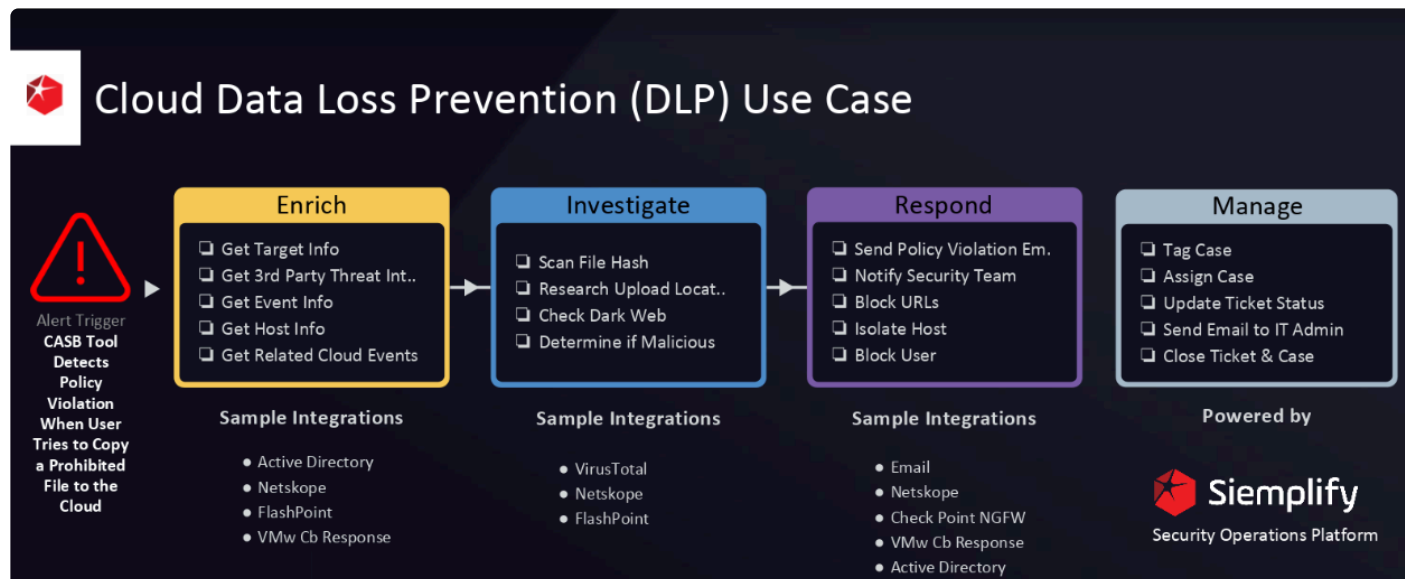
- Who: What entities are going to be affected (Scope)?
- Where: From where are you going to get the relevant information for the input parameters?
- How: How are you going to extract the needed information for the input parameters?
- What: What are the expected results?

Now, try and categorize the different actions into "logical" groups. A playbook typically has various stages it goes through:

- Enrichment
- Investigation
- Decision/Escalation
- Response/Remediation
- Manage/Logging (Communication and Ticketing)

Note: A playbook can be built from any subset of the above stages, and each can appear more than once (For example, you might want to take an immediate action before waiting for a manual decision from a user and then continue with the flow).

The following is a visual representation of a DPL flow.



2.2.4. Playbook Blocks – Identify Repeatable Logical Flows

To allow easier playbook building and management for large scale implementation, Simplify provides the unique ability to create modular sub-playbooks called playbook blocks. Just as software engineers build their code functions, playbook blocks represent a repeatable process, triage for example, and can include multiple actions, decision-making steps, inputs, and outputs. A block can be incorporated in multiple playbooks as a modular piece for faster playbook building, making them more readable for engineers. Simplify playbook blocks makes it easy to manage and modify large amounts of playbooks, since a change in a single block will affect all the playbooks using it. It is recommended that you take the time to consider the use of blocks in your overall development of automation (across playbooks, as these blocks can be used anywhere).

You should consider grouping up actions and logic into a block for:

Logical separation: Your playbook can be divided into logical sections, which can be grouped into Simplify playbook blocks. That way your overall playbook is going to be simpler and easier to understand (and easier to modify later on – as you only have to update a single block which can then impact all the playbooks using it).

Logical reuse: Sometimes, there is a specific sequence of actions that you might need multiple times. For example, updating of an external ticketing system or the enrichment of IOCs.

A good example for playbook block usage that covers both reasons above is 'Enrichment'. For starters, it usually consists of multiple actions and some decisions – all of which can be grouped into a single block, and therefore reduce clutter in the Playbook content (logic separation). In addition, enrichment is the most common stage an alert goes through. This 'Enrichment' playbook block can be used over and over in different playbooks. Furthermore, future updating of the enrichment process can be made in one place but impact on your automation everywhere.

2.2.5. Playbook Block Design

Before we dive into the process of designing a playbook block, you should be familiar with its capabilities. From the simplest of aspects, a playbook block is like a playbook. It is simply a sequence of actions you can use over and over in different places.

The major differences between playbook blocks and playbooks are:

- Playbook blocks have no trigger – as blocks run as part of a playbook and not on their own.
- Playbook blocks have an input and an output.

While designing the block you should keep in mind that it could be used in a vast variety of alerts (typically of different types as well). Constructing a generic block might be a little bit frustrating as you have to consider all edge cases, but it is going to be worth your while!

Standardized Input

If you are planning on using the block on multiple different alerts you might want to have some sort of standardized input. Let's say we are designing a block that quarantines hosts. It would be a good idea to have some way to indicate which hosts are designated for blocking, in which tools (if you have multiple relevant tools) and so on. A good example would be to 'mark' those entities with a specific flag (enrichment). The most common example here would be the 'Is Suspicious' property. In this example, the block would quarantine ANY hostname that is also suspicious, and it is up to you, in the playbook, to make sure you properly tag those hosts.

Standardized Output

The output is a little bit more straightforward. Let's say you have multiple enrichment tools running in the same block but they do not always return values – and you want to have proper results coming out of the block but you don't care which tool found results. In this case you should make sure that the results are standardized between all the different branches and that you get a single output out of the block. So for example it could be a malicious/benign indicator.

The "Is Suspicious" property of entities is a good example. Many actions mark an entity as suspicious based on different logic. You can do the same in your block, and thus convey a unified output for all playbooks using the block (naturally, you can use any other property instead – enrichment is considered output as well).

2.2.6. Design Tips

Playbook Blocks: Your main goal should be to work with blocks as much as possible. That being said, having actions outside blocks is legitimate so you should consider each case individually. Use blocks to have proper logic separation in your playbook, where applicable, or when you have a logic unit to be reused in different flows.

Manual action placement: Sometimes there is no way around some manual work or it is simply not available as an action yet. In these cases, it is recommended, if possible, to postpone the action as far as possible and have as much automatic flow as possible. This is a good place to start accepting the new approach of automation, where you can take more steps even if it is not always necessary, prior to a user's arrival, and have all data presented right away. In the long run it could free up a lot of analyst time.

Exit points: You are probably going to use multiple TI tools and sandboxes. It is all great and Simplify will take care of running it all, but it can still take time and stress other systems. It is recommended that after each enrichment attempt (if relevant) you check whether you can already establish a TruePositive or FalsePositive decision and move on right away, without completing the full investigative process. Obviously, if you need data from the rest of the action, for a report for example, make sure to run them.

Bottlenecks: Some actions, like sandbox scans, take time. You should design your automation to 'detonate' these actions, continue with whatever other flow you have, and only stop the playbook to 'wait' for the results once you have nothing else to do. For example, if you want to scan with 3 different sandboxes, you should trigger all 3 then wait for all 3 responses, instead of doing it one by one.

2.2.7. Summary of Implementation

By this stage should have a clear understanding of:

1. The data you are going to work with
2. The current flow taken by the professional team (Preferably as a sketch)
3. A mapping of the actions required by the manual flow to Simplify actions
4. An idea of logic separation/reuse for block building

A playbook in Simplify is built with the following hierarchy:

Action ⇒ Block ⇒ Playbook

If we take a second to compare playbook building to code writing the mapping would look like this:

1. Action ⇔ Basic function (Very specific/narrow)
2. Playbook Block ⇔ Complex function, made out of multiple basic functions
3. Playbook ⇔ Complete script/program made out of multiple functions of all types

In this guide, we are not going to discuss the creation of 'Actions'. To further learn about Actions, please refer to [Actions](#).

2.3. Build the Playbook Block

✿ Before you start this procedure, make sure you read up on the [Playbook Simulator](#)

1. Start by choosing the block you are going to develop. You should have its structure already designed as part of the complete playbook and by now you should roughly know what needs to be done as part of the block's automation.
2. Identify at least one of the alerts you are going to automate and simulate it. This new alert (remember the case ID) is going to serve you for the reminder of the guide as a test alert for the playbook's validation.
3. Head over to the 'Playbooks' tab and create a new block. Give it a proper name and turn on the simulation mode (Simulator).
4. Drag the action you want to add to the playbook block and place it in its appropriate place. A flow step may also be relevant here.
5. Configure the action using Placeholders/Expression-builder, entity, integration instances etc. More on this step later in the guide. Note, if you are missing parameters you have two options:
 - Result from a previous action – If this action does not exist yet, add that action to the block and work that action instead.
 - Information from the parent playbook – Add a new parameter to the block that represents this input and use it as a placeholder.
6. When you have finished building the playbook block, click Run to run the playbook block in simulation mode on an appropriate test alert.
7. Look at the step's result and make sure you are satisfied with it. If you are satisfied with it, hit "Pin Results" and change the action's state into simulation. You are done here and can return to the next step to continue designing your block.

Note: If your action enriches entities and you need it later on the block, you will have to manually add the relevant fields after you pin results.
8. If you are NOT satisfied with the step's result, you have multiple options:
 - Change the configuration
 - Replace the action with a different one
 - Click "Pin Results" and change the output to match your liking (relevant if you are querying a system that right now does not have data but you know how it should be structured)
9. Carry on repeating this procedure by adding new steps and running them in the simulator. Note that once you have pinned the results for a step, it will not run again in the Simulator. Make sure you switch back to normal mode if you want to rerun the Playbook block with fresh data.

2.3.1. Determine Playbook block output

Once you finish building a playbook block, in other words – when you are satisfied with the flow and it fulfills its purpose in your parent playbook, you should have: one or more actions/flows and possibly inputs. Now is a good time to consider the block's output. While it is true you are able to reference internal actions from a block in its parent playbook, it is generally considered ill-advised. Simplify recommends you divide a block's output into three (Note that you don't have to use any of these output suggestions; playbook block output is not mandatory):

Playbook Block output – A playbook block output is a simple string returned as a placeholder from the block. You can use it to return a simple boolean value, the number of results found in a query or any other 'string' result (ideally not a JSON, unless it is usable as a string)

Entity enrichment / context parameters – Playbook blocks can enrich entities, and this enrichment sticks with the entities throughout the case. That means that any enrichment, regardless of its origins (block or not) is going to stick and can be used as some sort of output. The most common field of enrichment to use is the "Is Suspicious" field. Many TI actions mark entities as suspicious, and you can use it in your playbook. Context parameters are variables you can set with actions in your block/playbook, and are stored at the alert/case level. You can have your playbook block modify these, and then use them from your playbook.

Action results (Not Recommended) You can always refer to direct actions from blocks. The issue here is that if you have flows in the block, you cannot know for sure which action actually ran, which is going to cause issues with placeholders. If you know with 100% confidence that an action ran, or you have a way to deal with unresolved placeholders, then you can use action results as output.

It is highly recommended to properly describe the block's expected input and output in the description fields.

2.4. Build Playbook

Playbook implementation is very similar to a Playbook Block implementation. The Playbook has a trigger which when set will cause the Playbook to run. Essentially, this is going to decide WHEN to apply the playbook (on which alert). Please refer to the [User Guide](#) for more information about triggers.

To build a playbook, make sure to turn on the Simulator and follow these steps:

1. Start building the Playbook with a trigger.
2. Drag and drop an action/flow/block – this time you have your blocks at your disposal and you can use these as well. Please note that blocks are either in simulation mode by definition or not. You cannot change their behavior from the playbook designer.
3. Configure action
4. Run action
5. Save results (save if satisfied, change and repeat if not)
6. Repeat steps above as necessary

(The short procedure above assumes that you have read the [Build the Playbook Block](#) section)

When you have finished designing your playbook and decide the Playbook is ready for production, you should turn off the simulation mode for each Playbook Block and Playbook so that they run in production mode.



Please note that Siemplify highly recommends that you use the playbook, still in simulation mode, as a training basis for your analysts and users. Get them familiar with the new automation and have them suggest changes before you take it into production officially.

2.5. Individual Features

So far we've covered the implementations at a high level. Now we will dive into details of how exactly how to do that, what to look for and how to utilize the different capabilities Simplify has to offer. Following are sections for each feature or tool Simplify has to offer for automation design and execution:

2.5.1. Placeholders and the Expression Builder

Placeholders are essential to the playbook. There is not much sense in writing automation that does not take into account the context of the alert at hand. Placeholders essentially allow you to 'inject' relevant information from the alert's context into your automation process. For instance, if you have a brute force alert, you can use a placeholder for the username and send an email to his manager (yet another placeholder) with details about the incidents (and more placeholders!)

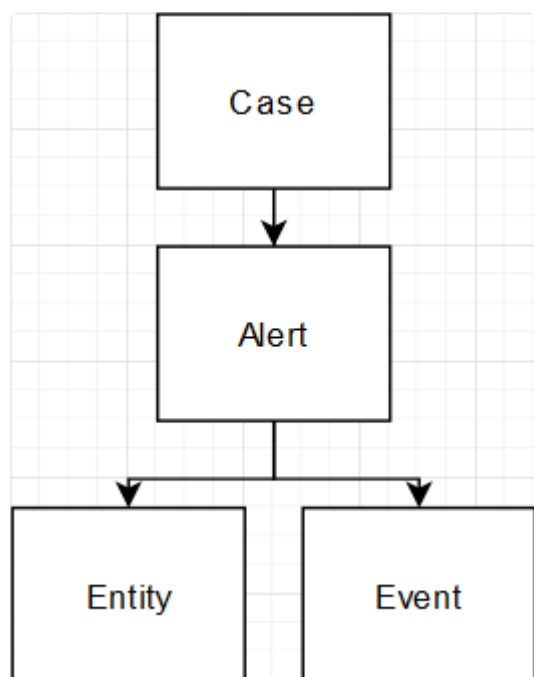
For this reason, it is extremely important that you take the time to understand placeholders in Simplify and how they resolve in real time. Also, take the time to play around with them and experiment on your own!

Simplify has six types of placeholders to offer, each representing its own portion of the context of the data. You should refer to the data model chart at the beginning of this guide for more information.

- Environment (+ Dynamic Environment Parameters)
- Case
- Alert (Note that a playbook runs on an alert)
- Entity
- Event
- Playbook

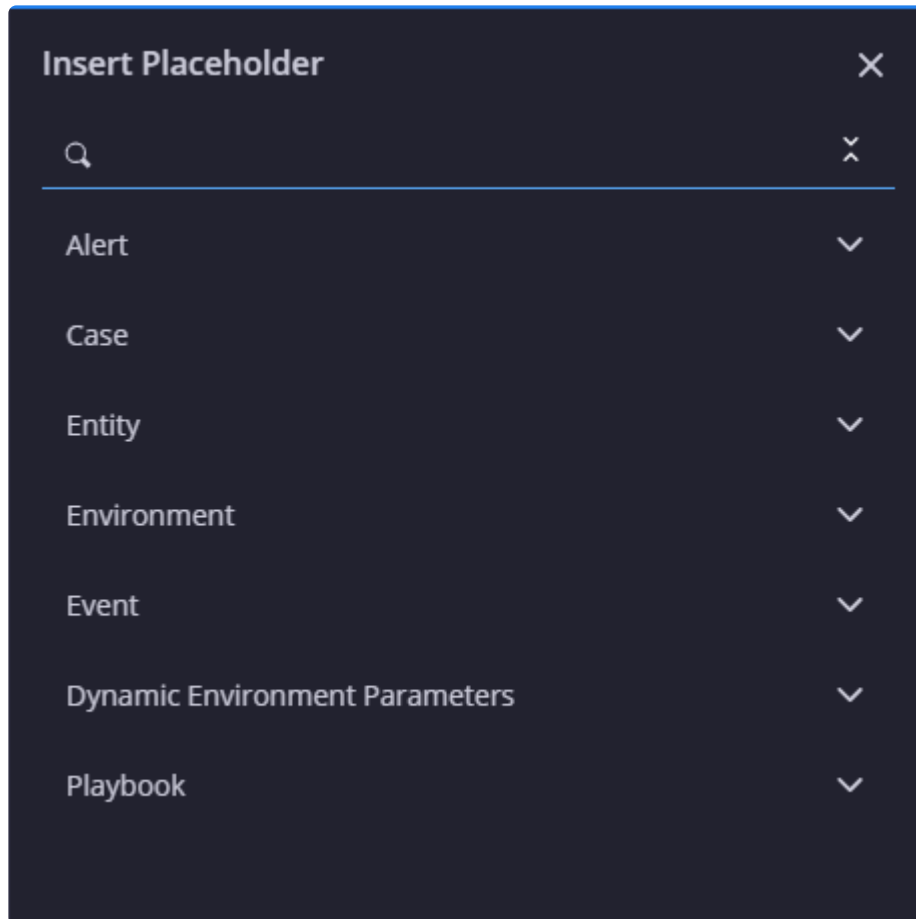
The placeholder types are ordered from the highest level of hierarchy to the lowest level. There are 'shared' properties that can be used/accessed from multiple locations, and their 'share' level is determined based on the above level of hierarchy.

Environment > Case> Alert> Entities/Events



And we have the playbook, which shares nothing with other playbooks (other than context parameters that

are saved on the alert/case level). Remembering the hierarchy is rarely needed for the implementation itself, but it is important to understand the data model you work with nonetheless! Here is a screenshot of the placeholder inserter with all 6 (+1) types of placeholders:



Simple Placeholders

Basic placeholder placement – The actual usage of placeholders in Siemplify is straightforward. It involves putting information inside square brackets '[]'. Your placeholder consists of two parts: type and key. The type would be one of the 6 types described above and the key will be the field you want to extract. For example, if you want to refer to the current case's ID, all you need to do is enter: [Case.Id]. The square brackets let Siemplify know it's a placeholder, then comes the type ("Case") and afterwards the key we want to extract ("Id"). This whole string, including the square brackets, will be replaced in real time with the case's ID.

Placeholders as part of a bigger input – You can use one or more placeholders simultaneously in one 'input' parameter by simply writing down your input and incorporating the placeholder syntax. For example, let's say we have an event field named 'username', and we want to write a comment to the case saying the user is involved in malicious activity. Our placeholder might look like this:

[Event.username] found to be involved in malicious activity.

Siemplify will know to only evaluate the term in the square brackets and leave the rest unchanged

Multiple placeholders – Simplify knows to evaluate placeholders independently over the input. That means you can use as many placeholders and free text as you want as part of your input parameter.

JSON placeholders and expression builder

The playbook type placeholder represents action results of the current running playbook. Actions, usually, have two results: 'simple' string result and a JSON result. The 'simple' result behaves exactly like the rest of the placeholder types. To let Simplify know you want a playbook-type placeholder, all you do is simply use the square brackets with the action's name and the script result name (it is automatically populated if you use the placeholder placement tool).

The JSON placeholder is different. You can still refer to it the same way you would refer to a normal placeholder, but the result will be a string representation of the JSON, mostly unusable as is. For that reason, Simplify has introduced the 'Expression builder', which is used to extract valuable information from the JSON.

The Expression Builder's UI looks like this:

Insert Placeholder

Json Sample

The sample below is for example purposes, actual data may vary.

```
{  "0": {    "Entity": "474B9CCF5AB9D72CA8A333889BBB3000",    "EntityResult": {      "malware": {        "created": "2014-10-20T23:19:00Z",        "family": [          "family [1]"        ],        "familyMembers": [          "familyMembers {1}"        ],        "hash": "0x474B9CCF5AB9D72CA8A333889B",        "md5": "0x474B9CCF5AB9D72CA8A333889B",        "origins": [          "origins {5}"        ],        "risk": "high",        "type": "md5"      }    }  }  }
```

Add Functions

- first
- last
- min
- max
- filter
- dateFormat
- count
- orderBy
- toLower
- toUpper
- replace
- distinct

Expression

```
| EntityResult.malware.hash
```

▶ Run

Results

```
0x474B9CCF5AB9D72CA8A333889BBB3000
```

Cancel

Insert

On the top left side of the screen, we have the JSON example. This is an example output JSON from the action result you have just referenced. This is not the real output of that action for the input you've supplied. It is strongly recommended to utilize the Playbook Simulator, run the referenced action and set its results to see the **actual** results here, instead of the example output.

To work with the Expression builder, all you need to do is browse through the JSON example, and click on the key you want to extract. In the screenshot above we've clicked on the "hash" key. Siemplify will automatically produce the relevant key-path you chose and present it in the "Expression" area below. At this point, you can click the "Run" button to make sure you've got the correct value. Note: Siemplify's placeholder logic will return a comma separated list of values whenever more than one possible result is found. In this example, if we had more entities, or simply a list of values for the "hash" field, the placeholder would resolve to a list of values separated by a comma.

For a more advanced manipulation of the JSON data, Siemplify has a list of functions at your disposal. You can find these functions on the right side of the screen. You can hover over each function for a short description and its signature (input parameters). To use these functions, all you need to do is click on it. Siemplify will automatically add that function, with a template of the parameters, to your expression.

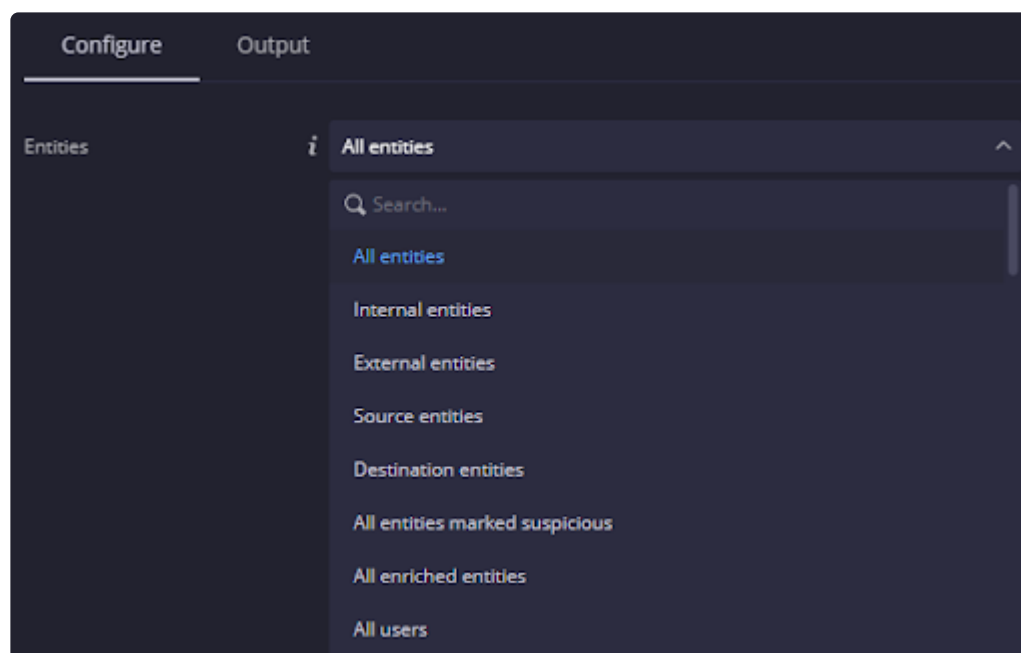
Please note that you can concatenate as many functions and key-path expressions together using the "|" (PIPE) symbol. For more information, please refer to [Using the Expression Builder](#).

2.5.2. Entities

Simplify is heavily based on entities and many actions run iteratively over them or simply consider them as input. Sometimes the input is explicit, like with a placeholder of type entity for example. Other times it is implicit, using the action's entity.

A straight forward example would be the enrichment type actions. These iterate over the entities and for each entity it reaches out to a third party product and provides information on that entity (Usually as properties on the entity object itself – Enrichment). Another example would be the response action. Let's say you want to quarantine a specific host, but you have two hosts in your alert. You will have to make sure you pass the correct entity to the action that quarantines entities, so you quarantine the correct host.

Simplify comes with a vast variety of different predefined scopes of entities. A small example can be seen in the screenshot below.



2.5.3. Conditions

The conditions allow you to direct your playbook in different directions depending on which criteria are met. So for example, if you are evaluating whether an email is suspicious or not, a true would send you down one path of further investigation and a false would send you down a path of closing the playbook. Without conditions you could not implement real logic in your playbook.

Simplify offers two types of conditions to be used in your design:

Multi Choice Question – Pose a question to the user and multiple options to choose from. This is used to ‘stop’ the automation and allow for the analyst to choose a specific route. The rest of the playbook could be either manual or automatic.

Automatic Condition – Evaluate one or more statements and choose a route based on the results. The structure of the statements resembles if/else-if/else structure. If the condition is NOT met, you attempt to evaluate the next statement. The first one that is met will dictate the route that is going to be taken. If no condition is met, the “Else” route is taken.

Both conditions will fork your playbook and allow for the execution of different flows. The only difference is the evaluator of the condition. The Multi Choice Question is evaluated by a user, while the Automatic Condition is evaluated by the Playbook service (Automatically).

Regardless of the condition you choose, the amount of actions or any other conditions you implemented, you can always ‘merge’ branches back into a single branch. To do that simply drag the action you want to be the first action of both branches into the last step of the branch you want to merge with.

Working with conditions in Simplify (Automatic conditions) is very much similar to working with actions. You are going to work closely with placeholders and entity Scope. As stated above, a condition in Simplify looks like a if/else-if/else statement. For each ‘if’ section you are required to specify a statement(s) condition. For example, [Entity.is_suspicious] = True, then if we have an entity marked as suspicious, it is evaluated as “True” and the corresponding branch is taken. It is highly unlikely that you will NOT use placeholders in a conditional statement, so it is a good time to make sure you are familiar with placeholders in Simplify.

You should pay close attention to placeholders of type event/entity. These can be plural and thus cause some issues with the planned evaluation of a condition. For instance, if you attempt to evaluate [Event.type] equals “login” and you have multiple events, the placeholder will resolve to a comma separated string and will never yield “True” as a result.

One more feature to consider in a condition is “Error Handling”. The next section deals with this in more details, but you should always remember some actions might fail and you need to have a plan ‘B’ for that. The condition step allows you to choose what happens in case of a failure of a previous action in the form of a “fallback” branch. The logic is simple: If a dependency action (one that its placeholder is used in the condition) has failed, take that branch and continue. You can use this to your advantage by escalating these alerts immediately, or by simply notifying your analysts about them.

2.5.4. Error Handling

This section deals with how to handle errors that happen in playbook actions, and which might inherently jeopardize the automation.

Siemplify offers a simple solution to this problem. Each action has a 'toggle' that dictates its behavior in case of a failure. A failed action in Siemplify is essentially a flag to the server. It could be an unprotected exception or a planned failure. In both cases, by default, the playbook service will stop the automation and wait for manual input from the user. However, if you want the playbook to continue automatically, for whatever reason, you can 'switch' the toggle off and prevent the playbook from waiting for input on failure of action.

You should be cautious when you choose to do so, as some other actions/conditions might rely on the result of the action (using placeholders for example). If the action is 'essential' to the playbooks flow, then you might want to reconsider skipping it in case it fails.

Fortunately, condition steps allow for a very easy way to 'catch' those errors and offers you, the designer of the automation, a way to overcome such issues with different logic. To do that all you have to do is set up a 'fallback' branch for a condition referring to the erroneous action. Let's discuss an example. Say you want to quarantine a host. This is, of course, a very sensitive action that must happen, otherwise you risk malware expansion. If the action failed, it means there is a good chance the host was not quarantined. In this case, it is reasonable that you don't want to wait for an analyst to notice the failure, nor can you simply skip the action and continue on with your life. What you should do is very simple! First, mark the action with the 'skip in case of failure' flag. Then, put a condition referring to the action result (does not have to be a real condition there, just the placeholder will do). Set the condition to fallback to one of the branches (depends on your logic) in case there is an error in one of the dependencies. If you only set that 'quarantine' action's placeholder, it is essentially a test whether the action failed or not. Now, on the fallback branch, put the logic you want to take when the quarantine fails (escalate, send messages, firewall rules etc.)

To sum up, we have two ways to deal with errors:

Hard: Action fails and playbook simply stops

Soft: Action skips. In this case we can separate it into two scenarios:

- Critical action – Need to introduce a condition to test whether the action failed and deal with it logically
- Uncritical action – Can allow it to simply skip and continue automation

In general it is recommended that for each action you take the time to consider these options and apply the correct piece of configuration to deal with it properly.

2.5.5. Environments

This section is mainly relevant for MSSPs, but can be utilized by enterprises as well. In general, Simplify allows for data separation between 'Environments'. Essentially, it means we have a complete separation between cases, credentials and integration configurations. This allows for the adaptation of multiple customers (Environment in Simplify) to reside in the same Simplify instance. Environments are also the highest level of hierarchy in Simplify's data model

When designing a block (or playbook), you are likely to use it in multiple playbooks and environments. Therefore it is recommended that you try and design the playbook block in a way that can be reused for as many environments as possible. Simplify supports configuration separation, and so you can configure different setting components and different integrations with different values for each environment, and the playbook service will know which one to run for you!

In addition, you can also utilize the Environment parameters. Simplify has a predefined set of parameters, to which you can add as many custom parameters as you wish. Each environment has these parameters and you can configure them separately. In your block, you can use an Environment placeholder to refer to these values, essentially allowing you to incorporate them in your logic.

Here's an example. Let's say you are an MSSP and you have many customers. Each customer has its own ticketing system (obviously, many customers use the same product, so it's not unique per customer). You could add an Environment parameter called "TicketingSystem" and configure it with the ticketing system's name. In the block, add a condition before communicating with the relevant ticketing system. This allows you to maintain a single block/playbook and offer the same protection to multiple customers.

2.5.6. Insights

Playbooks and automation in general has two main purposes:

Take action: You want some things to get done without intervention and as soon as possible.

Collect data for user: Help speed up investigation and allow for a more complete picture of the incident with minimal human effort (for investigation and human decision)

It is very easy to understand how automation solves the first issue. To help with the second issue you can use the “Insights”. Basically insights are a piece of text (can be HTML as well) that is presented on the case overview as the first thing you see when you open the case. A good playbook will expose the most important/relevant information to the case overview for the user to observe (be it actions that were taken and the user should be aware of or some important information).

There are two types of insights in Siemplify, the ‘General’ insight and the ‘Entity’ insight. Both will appear in the same place. The difference is just the ‘object’ to which they are tied to. Entity insight is related to an entity and will usually be added to the case overview by enrichment actions. Siemplify “Insights” integration allows you to create complicated entity insights on your own as well. And, of course, you can write your own actions to create entity insights (see our SDK documentation). The general insight is a little more straightforward. You give it a text and it dumps it to the case overview, as is.

You should take a look at the “Insights” integration (power-up) and its documentation to better understand your options and how it is recommended to create insights using Jinja2.

2.5.7. Simulate Alerts

Simplify offers a very easy way to test how certain configuration changes and playbook are going to affect real data. If you want to run tests, but you do not want these tests to reflect in your metrics/dashboards, all you need to do is simply 'simulate' an alert. Find that one alert you want to test with, hover over it and click the three dots menu. Select the "Simulate Alert" and choose an environment. A new (test) alert will be created for you, that will behave almost identically to a real alert. You can test playbooks and configurations on that alert, and simply close it afterwards, without affecting any other module.

Please note that actions you run on the simulated alert run for REAL. That means that if you choose to send an email, or quarantine a machine, this is actually going to happen. So please be cautious when running playbooks in general, but especially when you run tests.

2.5.8. How Playbooks work behind the scenes

There are two services responsible for playbook automation: Playbook service and Python service. The Python service actually serves all automation services in Siemplify. This is the service that executes the Python scripts and communicates with them to receive the results. The playbook service is responsible for the playbook actions (regular and async) and maintaining the action queue.

The playbook service has a limit of how many actions/flows it can run in parallel, which means that if you run a long action, or simply many many actions, you might quickly reach that limit and bottleneck the whole process. Fortunately, Siemplify has logic to alleviate such stress for some cases (not going to happen if you simply have too many actions)

If you encounter bottlenecks in your playbooks, be it a slow running playbook, or an overall long queue, you should consult Siemplify support. However, there are a few things you can do on your own. First, make sure that you don't run irrelevant action or use irrelevant scope for actions that run on entities. This will simply reduce the load. Next, if you run slow and long async actions, try to initiate the action at the beginning of the playbook, and wait for the result at the end. You could add more DPUs with Siemplify's help, if you expect high loads, or simply know to expect some slowdown on spikes of alerts. Depending on the available resources, you could also increase the amount of parallel playbooks.

3. Siemplify API

Click [Siemplify API](#) to see a full list of Siemplify API documentation.

4. SDK References

Siemplify's SDK was designed to help developers communicate and utilize various API functionalities. The SDK can be used from Siemplify's IDE in one of the three object types available in Siemplify:

- **[Actions](#)**

Actions are “stand alone” Python scripts that can be called, like functions, from either a playbook or manually by a user. A *SiemplifyAction* has (dynamic – based on alert data) data and configuration/parameters as input, and either returns a value or performs an action (or both).

- **Connectors**

Connectors are time-based Python scripts that run every predefined time interval with the purpose of creating new alerts in Siemplify (Ingesting new data). In principle, a connector is compared with a class in Python, as you create its definition once, but you can then instantiate multiple instances that can co-exist. A connector has only parameters as input, and nothing is dynamic (unless expressed in the logic).

- **Jobs**

Jobs are time-based Python scripts that run every predefined time interval. The difference between jobs and connectors is mainly their purpose. Connectors are responsible for creating new alerts in Siemplify, whereas jobs are usually used to sync data or maintenance.

The three objects above represent three types of automation (based on Python) that Siemplify utilizes and are presented in more detail in the next sections.

These 3 objects inherit from a more generic wrapper to Siemplify's API: “*siemplify*” and “*siemplifybase*”

NOTE: When using the SDK (from Siemplify's IDE), there is no need for authentication. The SDK itself is responsible for authentication.

4.1. Concepts & Tutorials

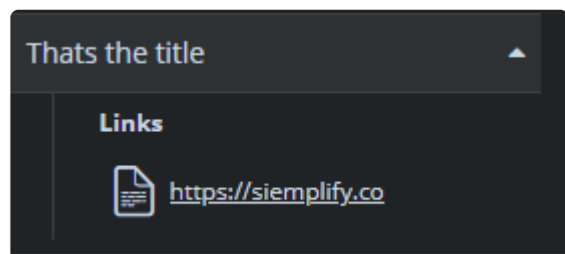
4.1.1. Actions

4.1.1.1. Action Results

Action results are viewable from an action context details, or case wall by clicking on the action block/entry and observing its results.

URL Links

Add a clickable *link* to the action result under a specific title (Usually the entity's identifier). Here's an example of a link added to the action result:



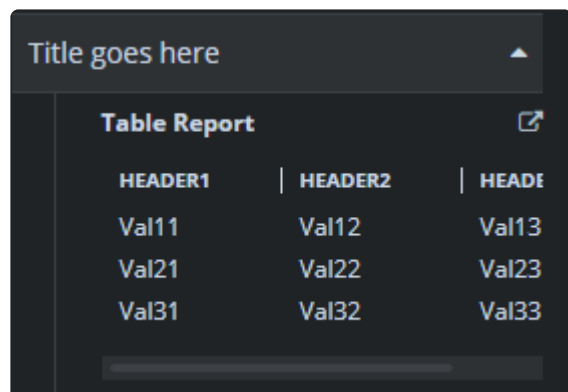
A common usage for this function is to display a link to a sandbox report or to an external service, where the analyst can watch a full report (Like VirusTotal or Cuckoo). However, you can also use this function to trigger a get request on one of your services, forcing the analyst to use specific parameters.

✿ Related Methods: [ScriptResult.add_link](#)

Data Tables

You can add a table display to the action result. This table, in turn, can be exported into a CSV that is downloaded to the local machine of the user.



Here's how it looks in Siemplify:

A screenshot of a dark-themed UI element. At the top, there is a title bar with the text "Title goes here" and a small upward-pointing triangle icon on the right. Below the title bar, the text "Table Report" is displayed in a bold font, followed by a square icon with an arrow pointing outwards. Below this, there is a table with three columns and four rows. The columns are labeled "HEADER1", "HEADER2", and "HEADER3". The rows contain values "Val11", "Val12", "Val13", "Val21", "Val22", "Val23", "Val31", "Val32", and "Val33".

HEADER1	HEADER2	HEADER3
Val11	Val12	Val13
Val21	Val22	Val23
Val31	Val32	Val33


To expand the result, simply click on the square with the arrow on the top right corner of table view. This pop-up will appear:

Title goes here



Header1	Header2	Header3
Val11	Val12	Val13
Val21	Val22	Val23
Val31	Val32	Val33

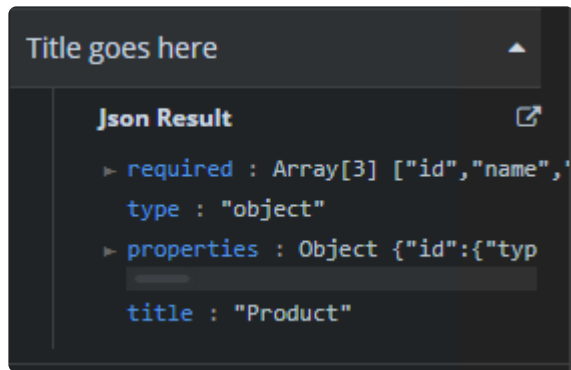
You can export this table to a CSV from this pop-up

 Related Methods: [ScriptResult.add_data_table](#)

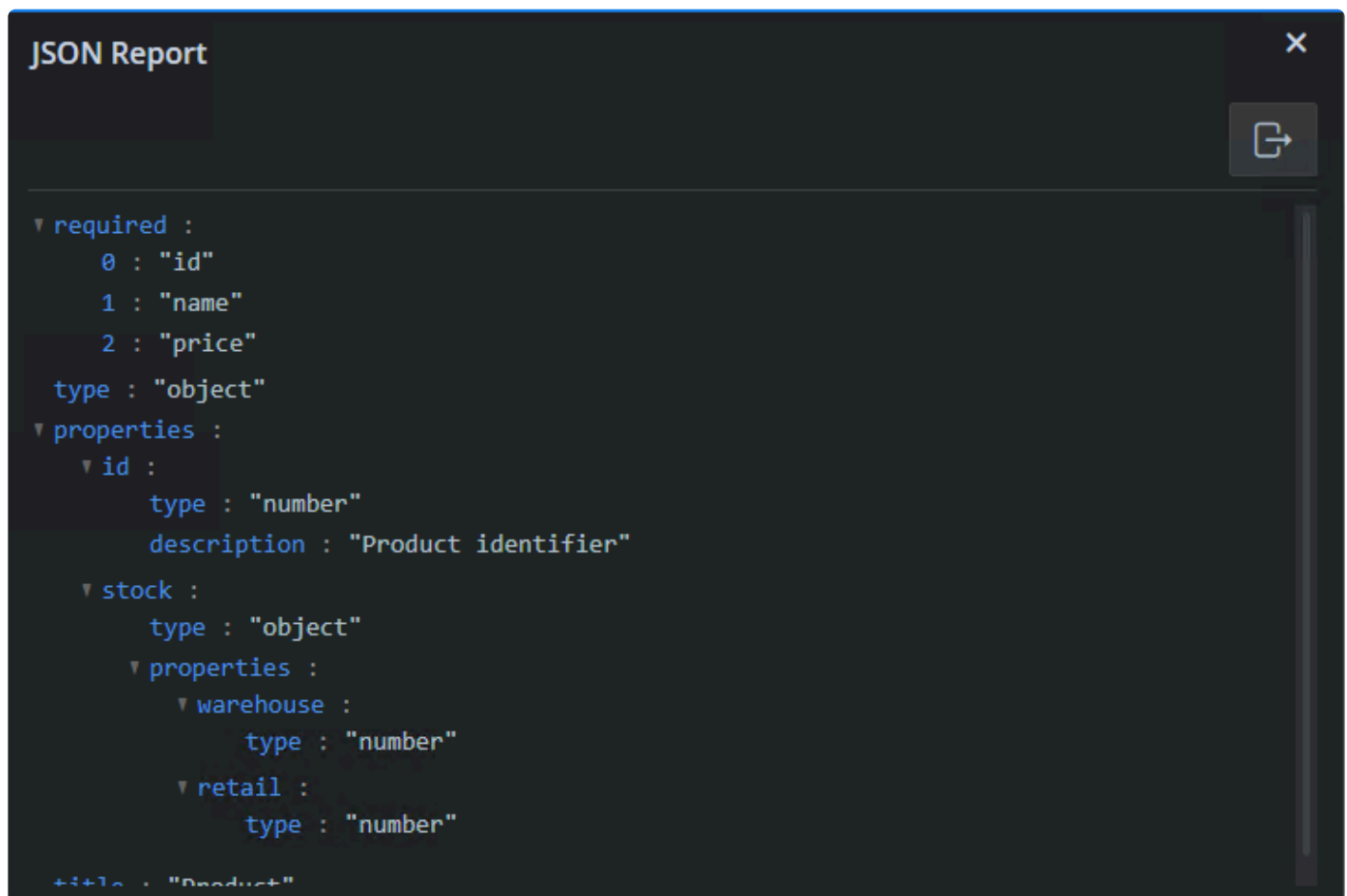
Json Results

You can add a [JSON](#) result, and view it in the system A common use for this function is displaying return values from API calls. Most third party integrations (and Simplify's API) return a JSON object when called.

Here's an example for a JSON viewer in Simplify (As an action result)



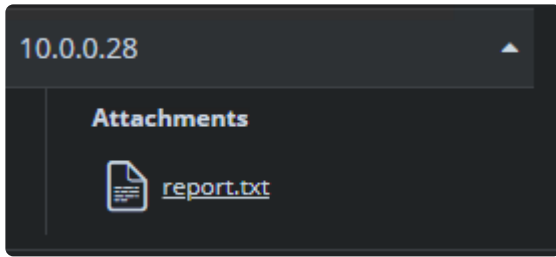
To expand the result, simply click on the square with the arrow on the top right corner of table view. This pop-up will appear:



Related Methods: [ScriptResult.add_json](#)

Attachments

You can add an attachment to the Action Result. The attachment will be displayed like an attached link or a table. Below is an image with example:



Here we have two HTML reports, one for each IP address. Clicking the link will download the HTML report to the local system of the user.

✿ Related Methods: [ScriptResult.add_attachment](#)

4.1.2. Integration Configuration & Script Parameters

Integration Configuration

Usually, each Integration will have Several configuration that needs to be set by the user, and used by the code.

The Configuration can be set by the user via the Marketplace screen, by clicking on the “Cog” icon on each installed integration card.



Related Methods: [Siemplify.extract_configuration_param](#), [SiemplifyAction.get_configuration](#)



It is advised to use the “extract_configuration_param” params, and not the raw “get_configuration” method

Script Parameters

Usually, each Action\Job\Connector script will require additional parameters, configured per Instance.

This can be configured by the Connectors screen, Jobs Screen, Playbook Screen, IDE Screen, or Manual Action Screen.



Related Methods: [SiemplifyJob.extract_job_param](#), [SiemplifyAction.extract_action_param](#), [SiemplifyConnectorExecution.extract_connector_param](#)

4.1.2.1. External Configuration Providers

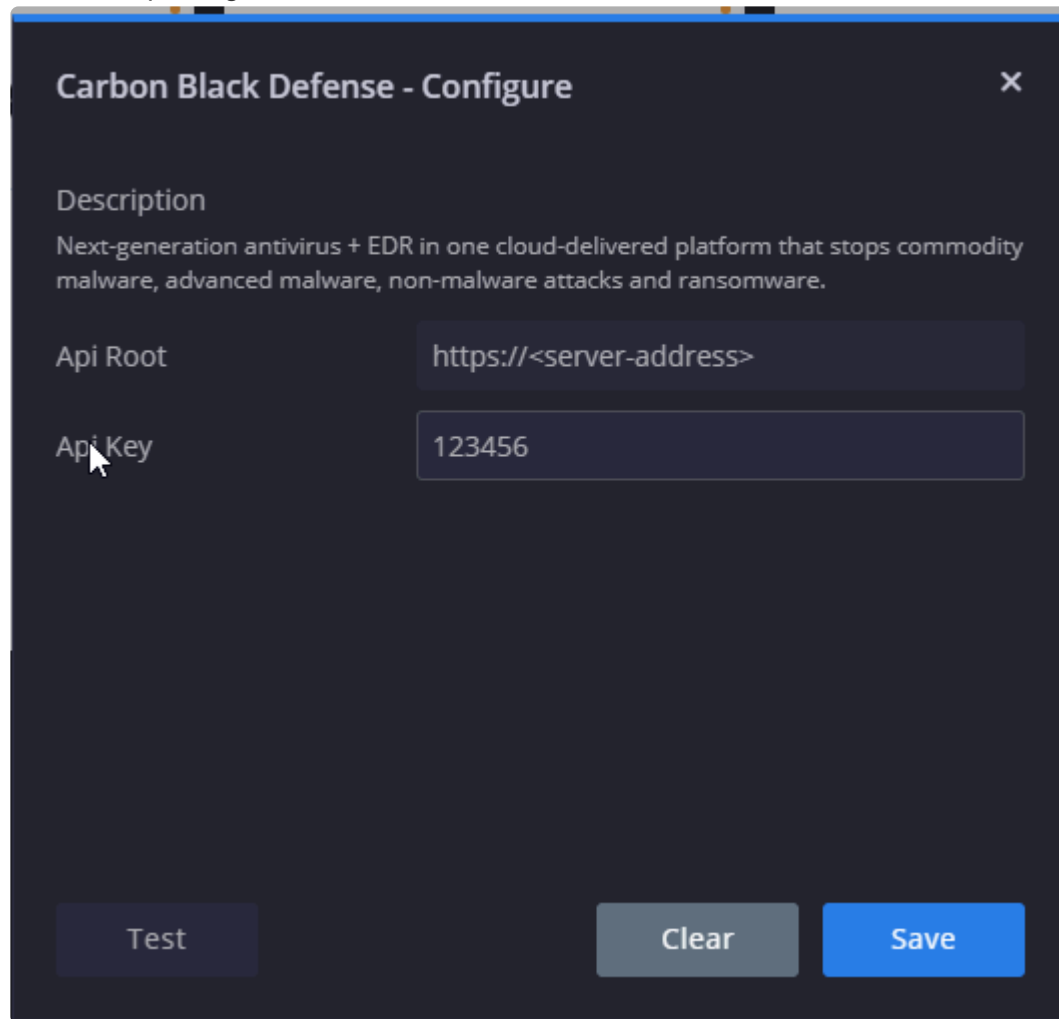
In the UI, when configuring an integration parameter, it is possible to input a placeholder that, on script runtime, will fetch the actual value, from an external source – usually a Credentials Vault Provider.

This is useful for:

- Dynamic changing credentials
- Organizations where the credentials are stored in a vault

For example:

Instead of placing Static credentials



The screenshot shows a configuration window titled "Carbon Black Defense - Configure". It contains a description of the product and two input fields. The "Api Root" field contains the placeholder "https://<server-address>" and the "Api Key" field contains the placeholder "123456". At the bottom, there are three buttons: "Test", "Clear", and "Save".

Carbon Black Defense - Configure

Description

Next-generation antivirus + EDR in one cloud-delivered platform that stops commodity malware, advanced malware, non-malware attacks and ransomware.

Api Root

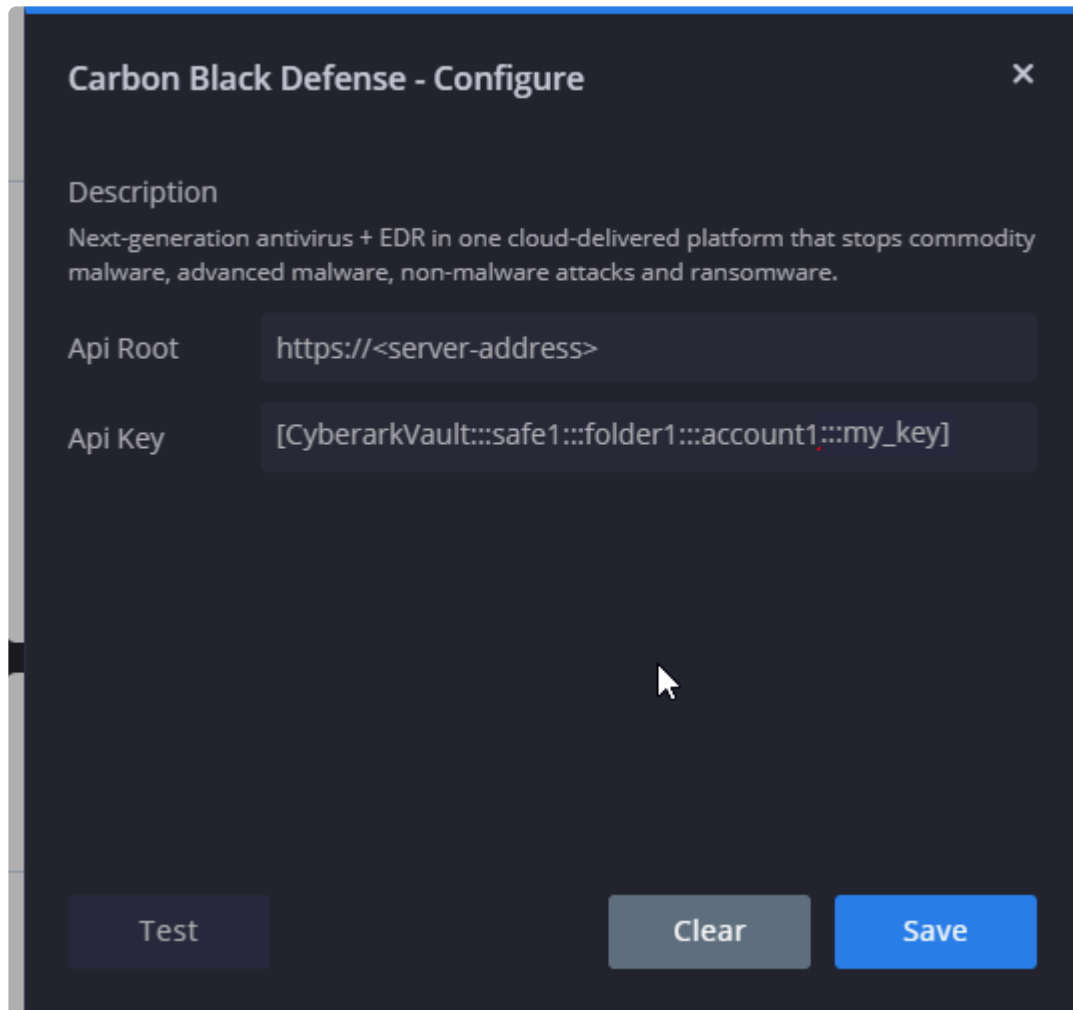
https://<server-address>

Api Key

123456

Test Clear Save

you can place



Carbon Black Defense - Configure [X]

Description
Next-generation antivirus + EDR in one cloud-delivered platform that stops commodity malware, advanced malware, non-malware attacks and ransomware.

Api Root:

Api Key:

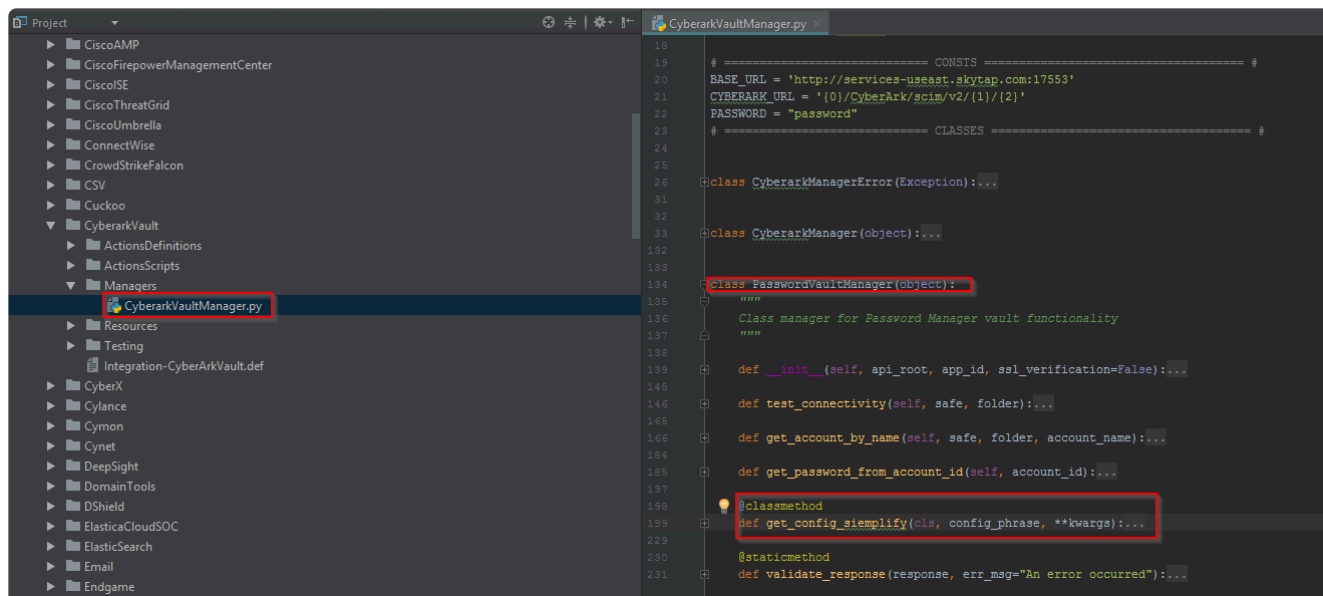
[Test] [Clear] [Save]

This will indicate to the siemplify system, to fetch the password param value from: a CyberarkVault provider (authenticate with pre-configured CyberArkVault integrations credentials), and fetch the value from the property “my_key” under “safe1”, “folder1”, “account1” and place it as the actual value of the Integration’s param “Password”

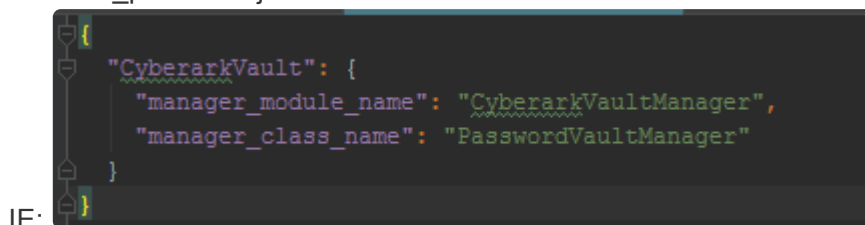
Placeholder Format:

[x1:::x2:::x3:::x4...]

- The external credentials provider format is a list of parameters, separated by “:::” inside Brackets. (There could be any number of params, as needed by the specific provider.)
 - The first param (ie x1) is the provider name:
1. Indicates the name of the external provider. The external provider will implement the fetching of the external data. It must be a Siemplify Integration, ie: CyberArkVault. **The name must match an installed Marketplace integration.**
 2. This integration must have a module, with a class, with a method called “get_config_siemplify” with the following signature:



- The name of the module + class, must be stated in file "...Bin\Scripting\PythonSDK\external_providers.json"



x2 to x4 and get_config_siemplify implementation:

```

@classmethod
def get_config_simplify(cls, config_phrase, **kwargs):
    """
    Static method for simplify get configuration third party apps mechanism
    :param config_phrase: {string} the config phrase from simplify
    :param kwargs: {dict} Arguments for manager instance
    :return:
    """
    api_root = kwargs['Password Vault Api Root']
    app_id = kwargs['Application ID']
    use_ssl = kwargs['Use SSL'].lower() == 'true'
    manager = cls(api_root, app_id, use_ssl)

    properties = config_phrase.split(":::")

    if len(properties) != 4:
        raise CyberarkManagerError("Invalid placeholder format: {}".format(config_phrase))

    safe, folder, account_name, prop = properties

    account = manager.get_account_by_name(safe, folder, account_name)

    if prop.lower() == PASSWORD:
        # If the desired property is password - replace it to content
        # as in CyberArk, the password is stored in Content field.
        prop = "Content"

    if prop not in account:
        raise Exception("Property {} doesn't exist in the given account.".format(prop))

    value = account[prop]

    return value

```

1. config_phrase = The original placeholder as inputted by the user, without the brackets or the first param (x1, aka external provider name), meaning "x2:::x3:::x4". So in our case "safe1:::folder1:::account1:::my_key"
2. **kwargs = the current integration (of the external provider) configuration, as configured and saved in Simplify (via the Marketplace UI)
3. In this example code, you can see the kwargs are used to define a 3rd party wrapper called cls, by which the "get_account_by_name" is called with provided safe, folder, account_name needed in order to fetch the actual value. Then, the result is sanitized to password values, and finally, the value itself is returned.

4.1.3. Custom Lists

✿ Related Methods: [Simplify.add_entities_to_custom_list](#), [Simplify.any_entity_in_custom_list](#), [Simplify.remove_entities_from_custom_list](#), [Simplify.get_existing_custom_list_categories](#), [Simplify.is_existing_category](#), [Simplify.get_custom_list_items](#)

The custom list is just a list of objects, that can be saved into Simplify's DBs, as a shared resource, fetched and queried by each script execution instance.

The custom list can be edited in settings screen's UI. This section deals with SDK functionalities for custom lists

Custom List item structure:


This object is defined in the `SiemplifyDataModel.py`

Param Name	Param Type	Possible Values
identifier	string	Any identifier whatsoever. Usually this identifier represents a possible entity in future alerts
category	string	Category from the Simplify settings
environment	string	Environment name from the Simplify settings. "*" refers to all environments





To create a *CustomList* object, do the following:

```
from SiemplifyAction import SiemplifyAction
from SiemplifyDataModel import CustomList
custom_list = CustomList(identifier="1.2.3.4", category="WhiteListed HOSTs", environment="")
```

When referring to the CustomList object in the future, this is its structure. Below, we can see an example of a single object in the settings screen. Every line represents a single CustomList object.

 SIEMPLIFY

[HOMEPAGE](#) [DASHBOARDS](#) [CASES](#) [PLAYBOOKS](#) [SEARCH](#) [REPORTS](#)

</>    

SETTINGS

Organization

Case Data

Advanced

Data Configuration

Ontology

Environments

Networks






Domains

Custom lists

Custom Lists

Define custom lists consisted of users, IPs and other entites.

Search...

Identifier	Category	Environment
1.2.3.4	BlackListed IPs	

4.1.4. Case Manipulation

This section focuses on SDK functions that help manipulate a case's state and data.

Name	Siemplify	SiemplifyAction	Description
close_case	NA	close_case	Closes the current case with the selected close reason and given comment
add_comment	add_comment	add_comment	Adds a comment to the current case's case wall
close_alert	NA	close_alert	Closes the current alert with the selected close reason and given comment
raise_incident	raise_incident	raise_incident	Changes the current case into an incident
assign_case	assign_case	assign_case	Assigns the current case to the selected analyst or group
add_entity_to_case	add_entity_to_case	add_entity_to_case	Creates a new entity in the case
add_attachment	add_attachment	add_attachment	Adds attachment to the case.

Case Metadata

This section focuses on SDK functions that manipulate case index and help maintain the system. Functions in this section are used to produce better search mechanism, KPIs and filters.

Methods:

Name	Siemplify	SiemplifyAction	Description
add_tag	NA	add_tag	Adds a new tag to the current case
change_case_stage	NA	change_case_stage	Sets the current case's stage to a specific stage
change_case_priority	change_case_priority	change_case_priority	Sets the current case's priority to a specific value
mark_case_as_important	mark_case_as_important	mark_case_as_important	Marks the current case with the 'importance triangle' sign

Data Retrieval

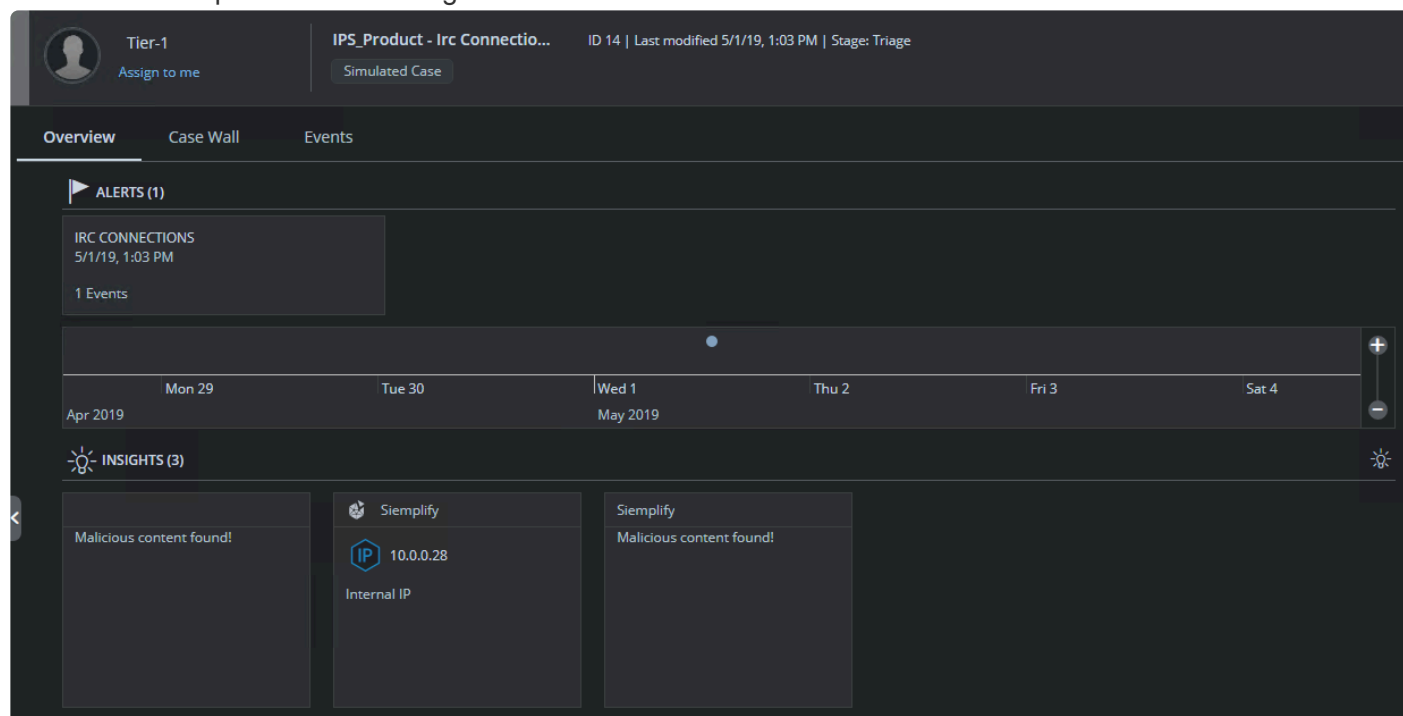
Name	Siemplify	SiemplifyAction
get_case_comments	get_case_comments	get_case_comments
get_alerts_ticket_ids_from_cases_closed_since_timestamp	NA	get_alerts_ticket_ids_from_cases_closed_since_timestamp
get_similar_cases	NA	get_similar_cases

4.1.4.1. Insights (General/Entity)

In Siemplify, insights are used to highlight important information that collected in the playbooks by various actions. For example, you might run a Threat Intelligence action to find out more information about a specific hash. The results can be seen in the case wall or the action results of the relevant action. However, if the report indicates that there is something malicious, you might want to expose the data to the Analyst outright, without further digging. This is easily done with insights.

An other example might be key enrichment values, such as ActiveDirectory's department enrichment, the amount of users that received a potentially malicious email, etc.

Here's an example of various insights:



In Siemplify, there are two types of insights:

- General insight
- Entity insight

The different between the insights' types is whether or not they revolve around entities. The middle insight in the picture is the only entity insight in the image, and it can be clearly seen what entity it refers to.

General Insight

✿ Related Methods: [create_case_insight](#)

Entity Insight

✿ Related Methods: [add_entity_insight](#)

! Note that this action creates a different insight for each entity in the action's scope.

4.2. API

4.2.1. SimplifyBase (SimplifyBase.py)

SimplifyBase class is never used directly. All three of Simplify's main SDK classes are inherited from SimplifyBase, so any function/property you see in this section is to be used from the main Simplify SDK component (SimplifyAction, SimplifyJob and SimplifyConnectorExecution)

4.2.1.1. fetch_timestamp

This function returns the timestamp.

```
fetch_timestamp(datetime_format=False, timezone=False)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
datetime_format	boolean	True/ False	True will return the datetime after converting the unixtime and False will simply return the datetime	No
timezone	boolean	True/ False	If True, siimplify will convert the time zone. [Not supported for DST]	No

Return Type

```
Datetime/int
```

Example:

Sample Code

```
from SiimplifyAction import SiimplifyAction  
sa = SiimplifyAction()  
result = sa.fetch_timestamp(datetime_format=True, timezone=False)
```

Result Behavior

The latest timestamp is fetched and is saved as `TIMESTAMP` file in the current directory.

Result Value

```
datetime.datetime(2019, 7, 16, 14, 26, 2, 26000)/1563276380
```


4.2.1.2. save_timestamp

This function saves the timestamp.

```
save_timestamp(self, datetime_format=False, timezone=False, new_timestamp=SiemplifyUtils.unix_now())
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
datetime_format	boolean	True/ False	True will return the datetime after converting the unixtime and False will simply return the datetime	No
timezone	boolean	True/ False	If True, siemplify will convert the time zone. [Not supported for DST]	No
new_timestamp	long	datetime	New timestamp as datetime to be saved.	No

Return Type

NoneType

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
sa = SiemplifyAction()
sa.save_timestamp(self, datetime_format=False, timezone=False, new_timestamp=SiemplifyUtils.unix_now())
```

Result Behavior

New timestamp will be saved as TIMESTAMP file in the current directory.

Result Value

None

4.2.1.3. fetch_and_save_timestamp

This function fetches the timestamp and saves the new timestamp to TIMESTAMP file in the current directory.

```
fetch_and_save_timestamp(self, datetime_format=False, timezone=False, new_timestamp=SiemplifyUtils.unix_now())
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
datetime_format	boolean	True/ False	True will return the datetime after converting the unixtime and False will simply return the datetime	No
timezone	boolean	True/ False	If True, siemplify will convert the time zone. [Not supported for DST]	No
new_timestamp	long	datetime	New timestamp as datetime to be saved. The default timestamp is the current time.	No

Return Type

```
Datetime/int
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
sa = SiemplifyAction()
sa.fetch_and_save_timestamp(self, datetime_format=False, timezone=False, new_timestamp=SiemplifyUtils.unix_now())
```

Result Behavior

The latest timestamp is fetched and is saved as TIMESTAMP file in the current directory.

Result Value

```
datetime.datetime(2019, 7, 16, 14, 26, 2, 26000)/1563276380
```

4.2.1.4. run_folder

This property returns the run folder based on the script name provided.

This folder can be used to store data/resources between script executions of the same script type.

```
run_folder()
```

Parameters:

N/A

Return Type

```
String
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
sa = SiemplifyAction()
sa.script_name("VirusTotal")
sa.run_folder()
```

Result Behavior

Folder named "VirusTotal" will be created and full path will be returned.

Result Value

```
jobs - /opt/siemplify/siemplify_server/Scripting/SiemplifyJob/{job_name}
actions - /opt/siemplify/siemplify_server/Scripting/SiemplifyAction/{action_name}
connectors - /opt/siemplify/siemplify_server/Scripting/SiemplifyConnectorExecution/{connector_name}
```

4.2.2. Simplify (Simplify.py)

The Simplify object inherits its properties from the SimplifyBase object and provides functionality for SimplifyAction and SimplifyJob.

4.2.2.1. add_Attachment

Related Concepts: [Case Manipulation](#)

This function adds an entry to the case wall with a file attachment (that can be then downloaded from the client into the user's local machine). The function does essentially the same thing as adding evidence (on the bottom of the case overview screen).

NOTE: To be able to upload a file to the case wall, you need to have the file available in the file system of the Siemplify server, or have it on a shared location accessible from the Siemplify server.

Here's a usage example:

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.add_attachment(r'C:/temp/investigation.txt', description='Deep investigation report by TIER3 team', is_favorite=True)
```

In this example, we will upload the “investigation.txt” from “C:/temp” on the local machine (the Siemplify server itself) to the case wall. A comment will be added to that entry on the case wall, with the string in the description. The “is_favorite” flag was set to “True”, and so this new entry will also be starred (favorite).

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
file_path	string	Any accessible file path	File path could be a remote location as well. You need read permissions to that file	Yes
case_id	string	A case ID to add the attachment to its case wall	The default is the current case	Yes
alert_identifier	string	Alert identifier string of the alert you want to associate the attachment with	The default is the current running alert	No
description	string	Any string	Default is empty string – Empty message with the attachment on the case wall	No
is_favorite	Boolean	True/False	Default is False	No

4.2.2.2. add_comment

✳ Related Concepts: [Case Manipulation](#)

This function adds a comment to the selected case.

```
siemplify.add_comment(comment, case_id, alert_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
comment	string	"This events in this alert seems suspicious"	Comments related to the case	Yes
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
add_comment = "This alert is important"
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.add_comment(comment, case_id, alert_identifier)
```

Result Behavior

The provided comment gets added to the case 234.

Result Value

None

4.2.2.3. add_entity_insight

✿ Related Concepts: [Insights](#)

This function adds entity insight to the selected entity identifier of the alert.

```
siemplify.add_entity_insight(domain_entity_info, message, case_id, alert_id)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
domain_entity_info	string	"8.8.8.8"	entity identifier	Yes
message	string	"This is Google DNS"	Message to add in the entity insight.	Yes
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
Boolean
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity = "8.8.8.8"
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.add_entity_insight(domain_entity_info=entity, message=message, case_id=case_id, alert_id=alert_identifier)
```

Result Behavior

The given message gets added as insight to the entity 8.8.8.8 of the given alert


```
identifier in the case 234.
```

Result Value

```
True [False if the insight is not added]
```

4.2.2.4. add_entity_to_case

✿ Related Concepts: [Case Manipulation](#)

This function adds entity insight to the selected entity identifier of the alert.

```
siemplify.add_entity_to_case(case_id, alert_identifier, entity_identifier, entity_type, is_internal, is_suspicious, is_enriched, is_vulnerable, properties, environment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes
entity_identifier	string	"8.8.8.8"	Unique entity Identifier	Yes
entity_type	string	"ADDRESS"	Entity type of the entity identifier.	Yes
is_internal	boolean	True/False	Internal: True, External: False	Yes
is_suspicious	boolean	True/False	suspicious: True, not suspicious: False	Yes
is_enriched	boolean	True/False	enriched: True, not enriched: False. Default is False .	Yes
is_vulnerable	boolean	True/False	vulnerable: True, not vulnerable: False. Default is False .	Yes
properties	dict	{"property": "value"}	Property of the entity.	Yes
environment	string	"Siemplify"	One of the defined environments in Siemplify.	Yes

Return Type

NoneType

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
case_id = "234"
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
entity = "8.8.8.8"
entity_type = "ADDRESS"
properties = {"property": "value"}
siemplify.add_entity_to_case(case_id=case_id,
                             alert_identifier=alert_identifier,
                             entity_identifier=entity,
                             entity_type=entity_type,
                             is_internal=True,
                             is_suspicious=False,
                             is_enriched=False,
                             is_vulnerable=False,
                             properties=properties,
                             environment=None)
```

Result Behavior

The entity with the provided information will be added to given alert with in the case 234.

Result Value

None

4.2.2.5. add_entities_to_custom_list

✿ Related Concepts: [Custom Lists](#)

This function gets a list of *CustomList* objects, representing lines in the *CustomList* settings table, and adds them to the table.

Each parameter should be explicitly specified – identifier, category and environment (all strings).

```
result = simplify.add_entities_to_custom_list([custom_list])
```

Parameters

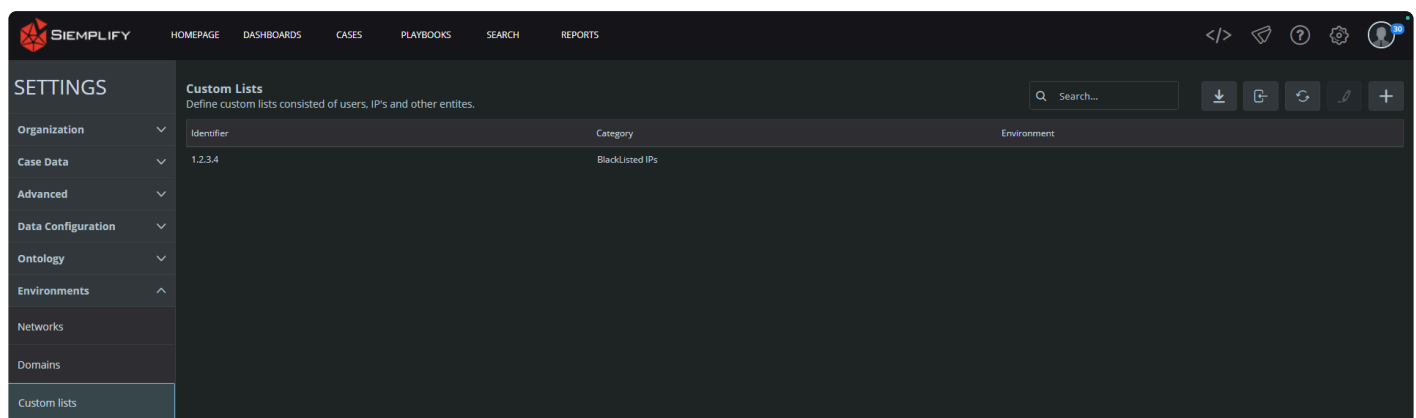
Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	"custom_list"	the custom list category	Yes

Return Type

```
List
```

Example:

Here's an example. Lets assume this is the state of the CustomList table prior to the function call:



Running `add_entities_to_custom_list` will result in a list of “CustomList” objects that represent configuration changes in the settings (added lines). Running the following code we get:

Sample Code

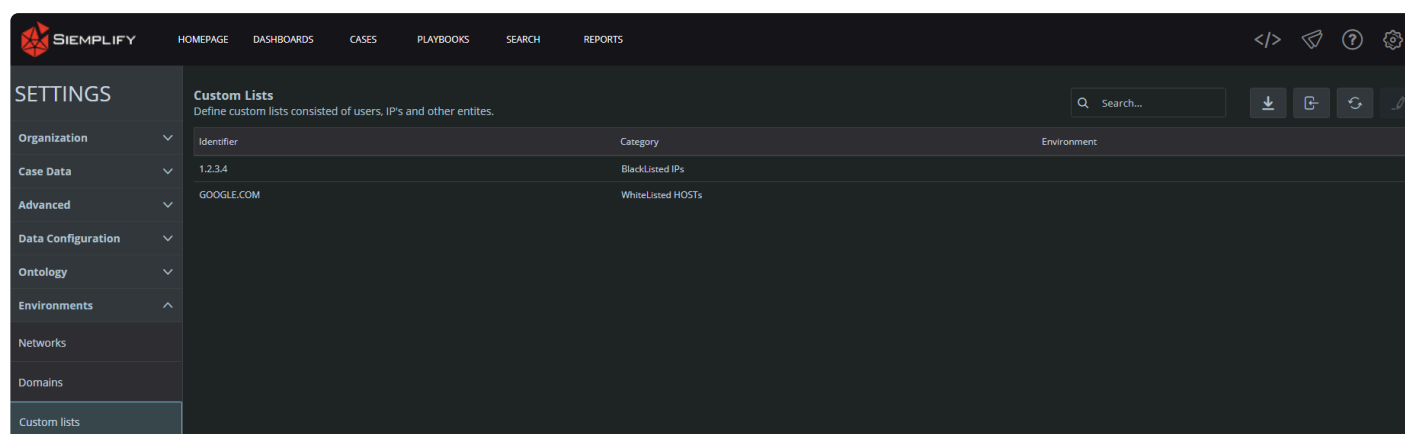
```
from SimplifyAction import SimplifyAction
```

```
from SimplifyDataModel import CustomList
custom_list = CustomList(identifier="GOOGLE.COM", category="WhiteListed HOSTs", environment="")

simplify = SimplifyAction()
result = simplify.add_entities_to_custom_list([custom_list])
```

Result Behavior

Entity is added to custom list category "WhiteListed HOSTs".



The screenshot shows the Simplify web interface. The top navigation bar includes links for HOMEPAGE, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, and REPORTS. On the left, a sidebar menu lists various settings categories: Organization, Case Data, Advanced, Data Configuration, Ontology, Environments, Networks, Domains, and Custom lists. The main content area is titled 'Custom Lists' and contains a table with columns for Identifier, Category, and Environment. The table lists two entries: '1.2.3.4' under 'BlackListed IPs' and 'GOOGLE.COM' under 'WhiteListed HOSTs'. A search bar and several action icons (download, share, refresh, edit) are located at the top right of the table.

Identifier	Category	Environment
1.2.3.4	BlackListed IPs	
GOOGLE.COM	WhiteListed HOSTs	



You can add multiple values from the *CustomList* or manipulate multiple lists with the same call. Simply add more *CustomList* objects to the list (each can have its own identifier, category and environment)

Result Value

[]

4.2.2.5.1. extract_configuration_param

✿ Related Concepts: [Integration Configuration & Script Parameters](#)

Get the value of an integration configuration parameter. Each integration has parameters that are part of its configuration (configured in the marketplace). This method allows extracting the value of a selected parameter from the integration's currently saved configurations.

```
param_value= simplify.extract_configuration_param(  
    provider_name,  
    param_name,  
    default_value=None,  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
provider_name	string	Any of the integration names in the marketplace	The name of the integration to get the parameter from its configuration	Yes
param_name	string	Any of the parameters names available in the integration's configuration	The name of the parameter to fetch	Yes
default_value		Any desired value	The default value of the parameter. The given value will be returned if the parameter was not set (if <code>is_mandatory</code> is set to <code>False</code>). Defaults to <code>None</code> .	No
input_type		Any valid python type	The type of the parameter. The returned value will be cast to the selected input type. Defaults to <code>str</code> .	No
is_mandatory	boolean	True/False	Whether the parameter is mandatory. If set to <code>True</code> and the parameter was not filled, an exception will be raised. Default to <code>False</code> .	No

print_value	boolean	True/False	Whether to output the fetched value of the parameter to the logs. Default to <i>False</i> .	No
--------------------	---------	------------	---	----

Return Type

As passed in input_type

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
param_value= siemplify.extract_configuration_param(
    "VirusTotal",
    "Api Key",
    default_value=None,
    input_type=str,
    is_mandatory=True,
    print_value=False)
```

Result Behavior

The value of the selected parameter will be returned, casted to selected type.

Result Value

123456

4.2.2.6. any_entity_in_custom_list

✿ Related Concepts: [Custom Lists](#)

Given a list of *CustomList* objects that represent lines from the *CustomList* settings, and returns True (Boolean) if any of them exists in the settings table. Otherwise, returns False (Boolean)
Each parameter should be explicitly specified – identifier, category and environment (all strings).

```
result_1 = simplify.any_entity_in_custom_list([custom_list_1])
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	List	"CustomList"	the custom list category	Yes

Return Type

```
Boolean
```

Example:

Here's an example. Below, the state of the system is presented:

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
custom_list_1 = CustomList(identifier="GOOGLE.COM", category="WhiteListed HOST
s", environment="")
custom_list_2  = CustomList(identifier="GOOGLE.COM", category="WhiteListed HOST
s", environment="Other_Environment")
custom_list_3  = CustomList(identifier="GOOGLE.COM", category="BlackListed HOST
s", environment="")
result_1 = simplify.any_entity_in_custom_list([custom_list_1]) # True
result_2 = simplify.any_entity_in_custom_list([custom_list_2]) # False
result_3 = simplify.any_entity_in_custom_list([custom_list_3]) # False
```

Result Behavior

In this example, `_result_1_` equals True. `_result_2_` and `_result_3_` are False com

paring to the system's state below.

System’s state

SIEMPLIFY

HOME PAGE

DASHBOARDS

CASES

PLAYBOOKS

SEARCH

REPORTS

</>

?

⚙

SETTINGS

Organization

Case Data

Advanced

Data Configuration

Ontology

Environments

Networks

Domains

Custom Lists

Custom Lists

Define custom lists consisted of users, IP's and other entites.

Q

Search...

↓

📄

↺

✎

Identifier	Category	Environment
1.2.3.4	BlackListed IPs	
GOOGLE.COM	WhiteListed HOSTs	

Result Value

True/False

4.2.2.7. assign_case

✿ Related Concepts: [Case Manipulation](#)

This function marks the current case with given alert identifier as important.

```
siemplify.assign_case(user, case_id, alert_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
assign_case	string	admin/@Teir1	Username such as "admin" or roles such as "@Tier1"	Yes
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
user = "admin"
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.assign_case(user=user, case_id=case_id, alert_identifier=alert_identifier)
```

Result Behavior

The case 234 gets assigned to *admin*.

Result Value

None

4.2.2.8. attach_workflow_to_case

This function attaches the workflow to current case.

```
siemplify.attach_workflow_to_case(workflow_name, cyber_case_id, indicator_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
workflow_name	string	workflow name	N/A	Yes
cyber_case_id	string	234	Unique case id	Yes
indicator_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	Unique alert identifier	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
workflow_name = "Workflow 234"
siemplify.attach_workflow_to_case((workflow_name=workflow_name, cyber_case_id=case_id, indicator_identifier=alert_identifier))
```

Result Behavior

```
Workflow 234 will be attached to case 234.
```

Result Value

```
None
```

4.2.2.9. change_case_priority

✿ Related Concepts: [Case Manipulation](#)

This function changes case priority.

```
siemplify.change_case_priority(priority, case_id, alert_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
priority	int	40/60/80/100	The priority mapping: {“Low”: 40, “Medium”: 60, “High”: 80, “Critical”: 100}	Yes
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
priority = 40
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.change_case_priority(priority=priority, case_id=case_id, alert_identi
er=alert_identifier)
```

Result Behavior

The case 234 priority gets changed to 40, which is mapped to low.

Result Value

None

4.2.2.10. create_case

This function creates Siemplify case with the alerts and events contained in the case_info dictionary

```
siemplify.create_case(case_info)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_info	dict	See example.	The details of the case to create. The dictionary keys must be in snake case.	Yes

Return Type

```
NoneType
```

Example

Sample code

[Example code](#)

Result Behavior

```
The case with the provided case data is created.
```

Result Value

```
None
```

4.2.2.11. end

This function ends the Simplify action and sends the action results to the Simplify system. This method is halting the action process, so no other code after the *end()* function will be executed.

```
end(output_message, result_value)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
output_message	string	Action completed	The message that will be displayed in Simplify.	Yes
result_value			The result value of the action (to be later used in other actions playbooks). For example, if an action is listing the users in Active Directory, then the action result might be the number of users found. Usually a boolean value will be passed, indicating whether the action succeeded or not.	Yes
execution_state	int	0/1/2/3	Indicator for the current action's state, 0 = completed, 1 = in progress, 2 = failed, 3 = timed out (states can be found in ScriptResult module). Mainly used in async actions for marking whether the action has completed or not. Default to 0	No

Return Value

None

Errors

If the **end** function is not called, the script will throw the following error.

```
Script did not return expected data. Did you call build_result/end_script?  
Check DebugOutput for details
```


Example

Sample code

```
from SiemplifyAction import SiemplifyAction
from ScriptResult import EXECUTION_STATE_COMPLETED
siemplify = SiemplifyAction()
output_message = "Display message when action is done."
result_value = True
siemplify.end(output_message, result_value, EXECUTION_STATE_COMPLETED)
```

Result

```
output_message
```

4.2.2.12. end_script

This function is **deprecated**.

4.2.2.13. get_case_comments

✿ Related Concepts: [Case Manipulation](#)

This function gets the comments from the provided case.

```
get_case_comments(case_id)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	12314	Unique case id	No

Return Type

```
List
```

Example

Sample code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
simplify.get_case_comments(case_id)
```

Result Behavior

All comments belonging to case will be fetched.

Result Value

```
[
    {
        u 'comment': u 'Test',
        u 'case_id': 10085,
        u 'is_favorite': False,
        u 'alert_identifier': None,
        u 'creator_user_id': u 'Admin',
        u 'type': 5,
```

```
    u 'id': 1,  
    u 'modification_time_unix_time_in_ms': 1563272078332L  
  }, {  
    u 'comment': u 'jhfksth',  
    u 'case_id': 10085,  
    u 'is_favorite': False,  
    u 'alert_identifier': None,  
    u 'creator_user_id': u 'Admin',  
    u 'type': 5,  
    u 'id': 2,  
    u 'modification_time_unix_time_in_ms': 1563272079941L  
  }, {  
    u 'comment': u 'kjfhshdm',  
    u 'case_id': 10085,  
    u 'is_favorite': False,  
    u 'alert_identifier': None,  
    u 'creator_user_id': u 'Admin',  
    u 'type': 5,  
    u 'id': 3,  
    u 'modification_time_unix_time_in_ms': 1563272080598L  
  }  
]
```

4.2.2.14. get_existing_custom_list_categories

✿ Related Concepts: [Custom Lists](#)

This function returns a list object of all the categories in the *CustomList* settings irrespective of Environments. It simply returns all the values available.

```
result = simplify.get_existing_custom_list_categories()
```

Parameters

N/A

Return Type

List

Example:

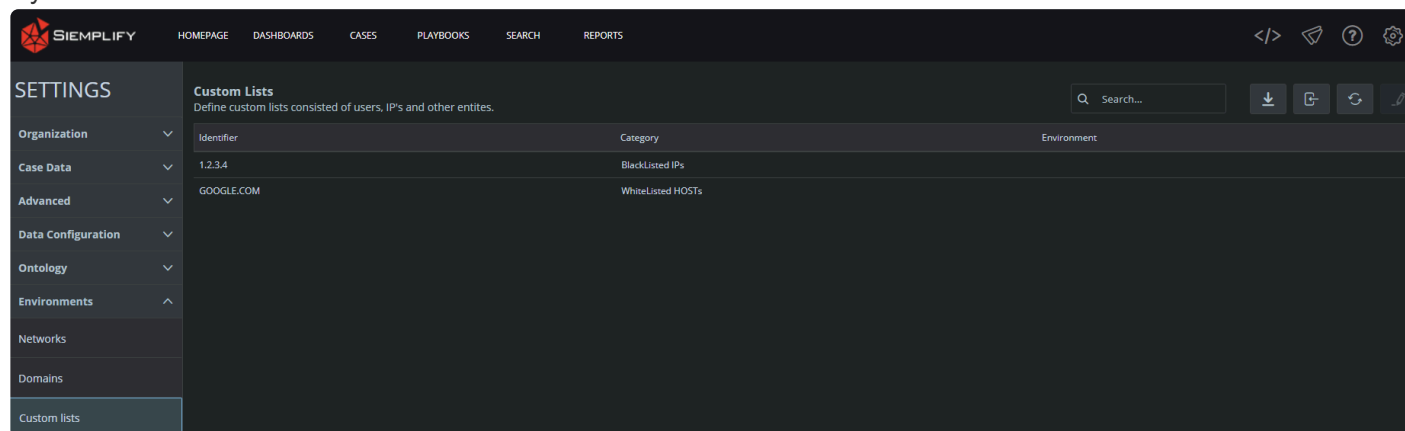
Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify.get_existing_custom_list_categories()
```

Result Behavior

A list of all existing custom list is returned.

System's state



The screenshot displays the Simplify application's settings page, specifically the 'Custom Lists' section. The interface is dark-themed. On the left, a sidebar lists various settings categories: Organization, Case Data, Advanced, Data Configuration, Ontology, Environments, Networks, Domains, and Custom lists. The 'Custom Lists' category is selected. The main content area shows a table with the following data:

Identifier	Category	Environment
1.2.3.4	BlackListed IPs	
GOOGLE.COM	WhiteListed HOSTs	

At the top of the main content area, there is a search bar and several action icons (download, copy, refresh, etc.). The header of the application shows the 'SIMPLIFY' logo and navigation links: HOMEPAGE, DASHBOARDS, CASES, PLAYBOOKS, SEARCH, and REPORTS.

Result Value

```
["BlackListed IPs", "WhiteListed HOSTs"]
```

4.2.2.15. is_existing_category

✿ Related Concepts: [Custom Lists](#)

Given a category name, this function returns True (Boolean) if the exact *category name* string is defined as a category in the *CustomList* settings.

This function does not take Environment into account – It simply returns True if it exists at all, otherwise, False.

```
siemplify.is_existing_category("WhiteListed HOSTs")
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category	string	"BlackListed IPs"	the custom list category	Yes

Return Type

```
Boolean
```

Example:

Sample Code 1

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.is_existing_category("WhiteListed HOSTs")
```

Sample Code 2

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.is_existing_category("SpecialHosts")
```

Result Behavior

The result in Sample Code 1 returns True and result in the Sample Code 2 returns False.

System's state

SIEMPLIFY

HOME PAGEDASHBOARDSCASESPLAYBOOKSSEARCHREPORTS

</>📧?⚙️

SETTINGS

Organization▼Case Data▼Advanced▼Data Configuration▼Ontology▼Environments^NetworksDomainsCustom lists

Custom Lists

Define custom lists consisted of users, IP's and other entites.

Q

Search...

⬇️🔗🔄✎

Identifier	Category	Environment
1.2.3.4	BlackListed IPs	
GOOGLE.COM	WhiteListed HOSTs	

Result Value

True/False

4.2.2.16. mark_case_as_important

✿ Related Concepts: [Case Manipulation](#)

This function marks the current case with given alert identifier as important.

```
siemplify.mark_case_as_important(case_id, alert_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.mark_case_as_important(case_id=case_id, alert_identifier=alert_identifier)
```

Result Behavior

The case with the provided alert identifier will be marked as important.

Result Value

```
None
```

4.2.2.17. raise_incident

✿ Related Concepts: [Case Manipulation](#)

This function raises the given case with the alert identifier as incident.

```
siemplify.raise_incident(case_id, alert_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	234	Unique Case Identifier	Yes
alert_identifier	string	ad6879f1-b72d-419f-990c-011a2526b16d	N/A	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
alert_identifier = "ad6879f1-b72d-419f-990c-011a2526b16d"
case_id = "234"
siemplify.raise_incident(case_id=case_id, alert_identifier=alert_identifier)
```

Result Behavior

The case 234 will be raised as incident.

Result Value

```
None
```

4.2.2.18. remove_entities_from_custom_list

✿ Related Concepts: [Custom Lists](#)

This function gets a list of *CustomList* objects, representing lines in the *CustomList* settings table, and removes them from the table.

Each parameter should be explicitly specified – identifier, category and environment (all strings).

```
result = simplify.remove_entities_from_custom_list([custom_list])
```

Parameters

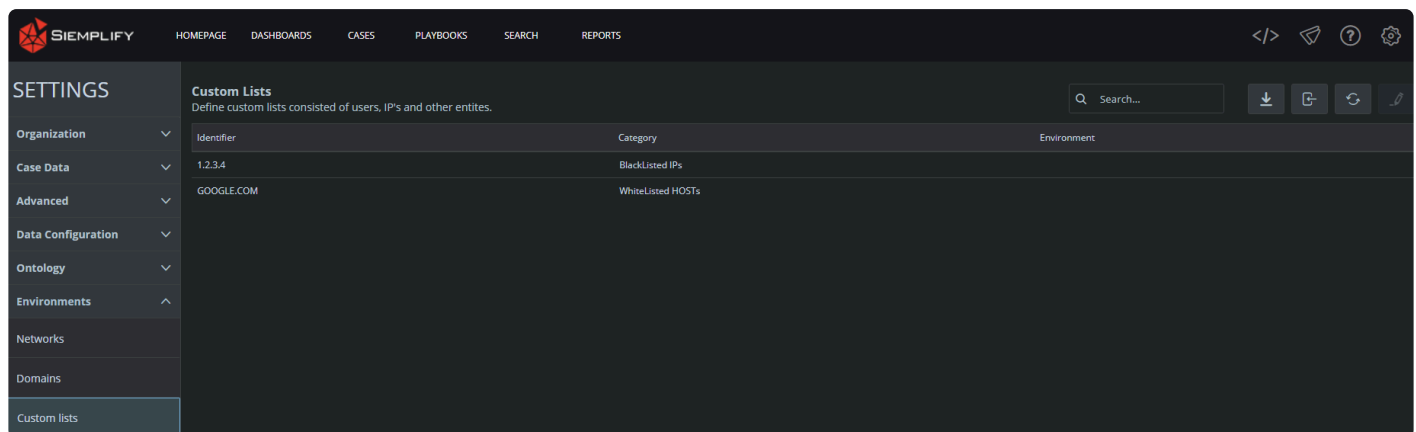
Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	the custom list category		Yes

Return Type

```
List
```

Example:

Here's an example. Let's assume this is the state of the CustomList table prior to the function call:



Running *remove_entities_from_custom_list* will result in a list of “CustomList” objects that represent configuration changes in the settings (removed lines). Running the following code we get:

Sample Code

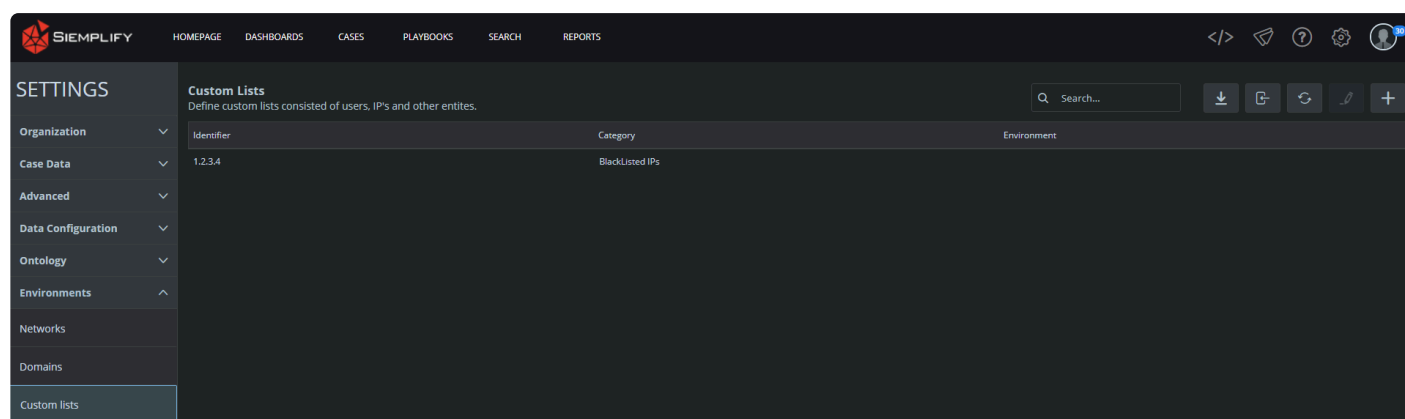
```
from SimplifyAction import SimplifyAction
```

```
from SiemplifyDataModel import CustomList
custom_list = CustomList(identifier="GOOGLE.COM", category="WhiteListed HOSTs", environment="")

siemplify = SiemplifyAction()
result = siemplify.remove_entities_from_custom_list([custom_list])
```

Result Behavior

The custom category "WhiteListed HOSTs" is removed.



Result Value

[]



You can remove multiple values from the *CustomList* or manipulate multiple lists with the same call. You can add more *CustomList* objects to the list (each can have its own identifier, category and environment)

4.2.2.19. update_entities

This function adds the new entities to the alert.

```
siimplify.update_entities(updated_entities)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
updated_entities	list	["GOOGLE.COM", "8.8.8.8"]	List of entities to add to the case.	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiimplifyAction import SiimplifyAction
siimplify = SiimplifyAction()
new_entities = ["GOOGLE.COM", "1.2.3.4"]
siimplify.update_entities(updated_entities=new_entities)
```

Result Behavior

The selected alerts via the scope gets new entities added if they are not present in the alert.

Result Value

```
None
```

4.2.3. **SimplifyAction (SimplifyAction.py)**

The SimplifyAction object inherits its properties from the Simplify object, which inherits its properties from the SimplifyBase object.

SimplifyBase = Grandfather

Simplify = Father

SimplifyAction = Child

4.2.3.1. add_attachment

✿ Related Concepts: [Case Manipulation](#)

This function gets a list of custom list items from category and entities list. This function returns a list of custom list item objects.

```
result = siemplify.add_attachment(file_path, case_id, alert_identifier, description, is_favorite)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
file_path	string	"C:\Program Files (x86)\Google\Chrome\Application\chrome_proxy.exe"		Yes
case_id	string	234	Unique case id	No
alert_identifier	string	12345	Unique alert identifier.	No
description	string	"The description for the file"		No
is_favorite	boolean	True/False		No

Return Type

```
String
```

Example:

Input: Explicitly, File path, description and is_favorite. Implicitly, case_id and alert_identifier.

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.add_attachment("C:\Program Files (x86)\Google\Chrome\Application\chrome_proxy.exe", case_id="234", alert_identifier=None, description=None, is_favorite=True)
```

Result Behavior

The file mentioned in the file path will be attached to case id 234 and attachment id will be returned.

Result Value

5 [The attachment id]

4.2.3.2. add_comment

✿ Related Concepts: [Case Manipulation](#)

This function adds a comment to the current case's case-wall. This function does the same thing as a user typing down a comment and saving it on the case wall.

```
siemply.add_comment(comment=comment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
comment	string	Any string could be used here		Yes
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplyAction import SiemplyAction
siemply = SiemplyAction()
comment = "Ran some tests on the hash and it seems fine"
siemply.add_comment(comment=comment)
```

Result Behavior

The specified comment is added to the current case.

Result Value

```
None
```

4.2.3.3. add_entity_to_case

✿ Related Concepts: [Case Manipulation](#)

This function adds an entity to the case.

```
add_entity_to_case(entity_identifier, entity_type, is_internal, is_suspicious, is_enriched, is_vulnerable, properties, case_id, alert_identifier, environment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com		Yes
entity_type	string	0: "SourceHostName" 1: "SourceAddress" 2: "SourceUserName" 3: "SourceProcessName" 4: "SourceMacAddress" 5: "DestinationHostName" 6: "DestinationAddress" 7: "DestinationUserName" 8: "DestinationProcessName" 9: "DestinationMacAddress" 10: "DestinationURL" 11: "Process" 12: "FileName" 13: "FileHash" 14: "EmailSubject" 15: "ThreatSignature" 16: "USB" 17: "Deployment" 18: "CreditCard" 19: "PhoneNumber" 20: "CVE" 21: "ThreatActor" 22: "ThreatCampaign" 23: "GenericEntity" 24: "ParentProcess" 25: "ParentHash"		Yes

		26: "ChildProcess" 27: "ChildHash" 28: "SourceDomain" 29: "DestinationDomain" 30: "IPSet"		
is_internal	boolean	True/False	True: Internal, False: external	Yes
is_suspicious	boolean	True/False	True: Suspicious, False: Not suspicious	Yes
is_enriched	boolean	True/False	True: Enriched, False: Not enriched. The default value is False.	Yes
is_vulnerable	boolean	True/False	True: Vulnerable, False: Not vulnerable. The default value is False	Yes
properties	dict	{"Property1": "PropertyValue", "Property2": "PropertyValue2"}		Yes
case_id	string	12345	Unique case identifier. The case_id value defaults to None.	No
alert_identifier	string	123123	Unique alert identifier. The alert_identifier value defaults to None.	No
environment	string	Siemplify, Apple	Environment name as defined in Siemplify system. The environment value defaults to None	No

Return Type

NoneType

Error

If there is an existing Entity, Siemplify will throw the following error.

```
500 Server Error: Internal Server Error for url: https://localhost:8443/api/external/v1/sdk/CreateEntity?format=snake: {"ErrorMessage": "Cannot add entity [Identifier:Entities Identifies - Type:siemplify.parameters[] to alert [MONITORED MAILBOX <FREETRIAL@SIEMPLIFY.CO>_633997CB-D23B-4A2B-92F2-AD1D350284FF] in case [30703] because the entity already exists there.\"}
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.add_entity_to_case(entity_identifier, entity_type, is_internal, is_suspicious, is_enriched, is_vulnerable, properties, case_id, alert_identifier, environment)
```

Result Behavior

This function will add a new entity to the case if it is not present in the case.

Result Value

None

4.2.3.4. add_alert_entities_to_custom_list

This function gets a category name (From CustomLists in the Simplify settings) and returns a list of objects of type *CustomList* (Refer to the SimplifyDataModel for more info) for any of the entities in the scope that were added to the chosen category.

NOTE: The Environment is added implicitly from the alert's environment!

```
result = simplify.add_alert_entities_to_custom_list("WhiteListed HOSTs")
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	"CustomList"	the custom list category	Yes

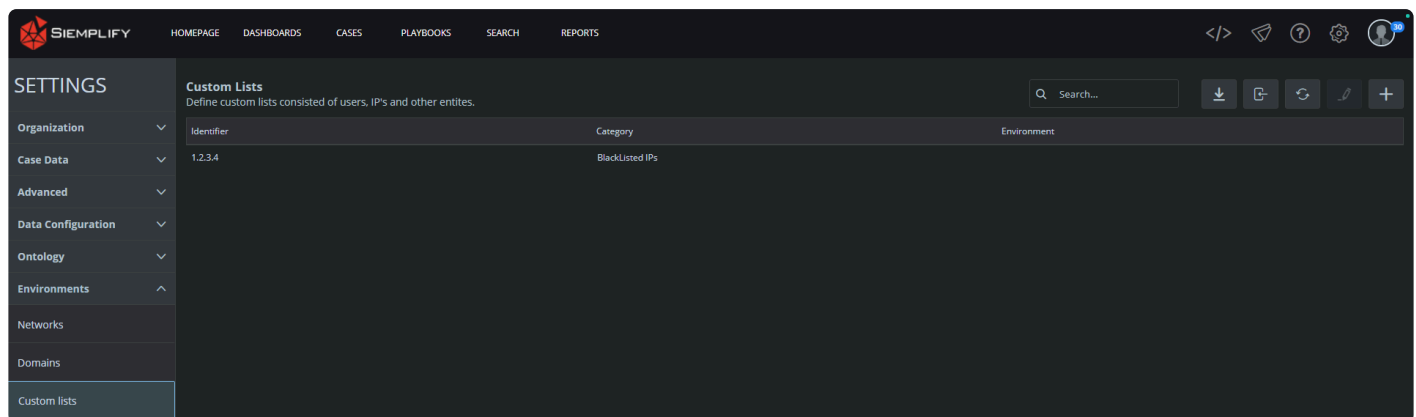
Return Type

```
List
```

Example:

Input: Explicitly, category_name. Implicitly, entities via scope.

Let's assume this is the state of the CustomList table prior to the function call, and let's assume the scope of the action has a single entity, "GOOGLE.COM"



Running add_alert_entities_to_custom_list will result in a list of "CustomList" objects and a configuration change in the settings. Running the following code we get:

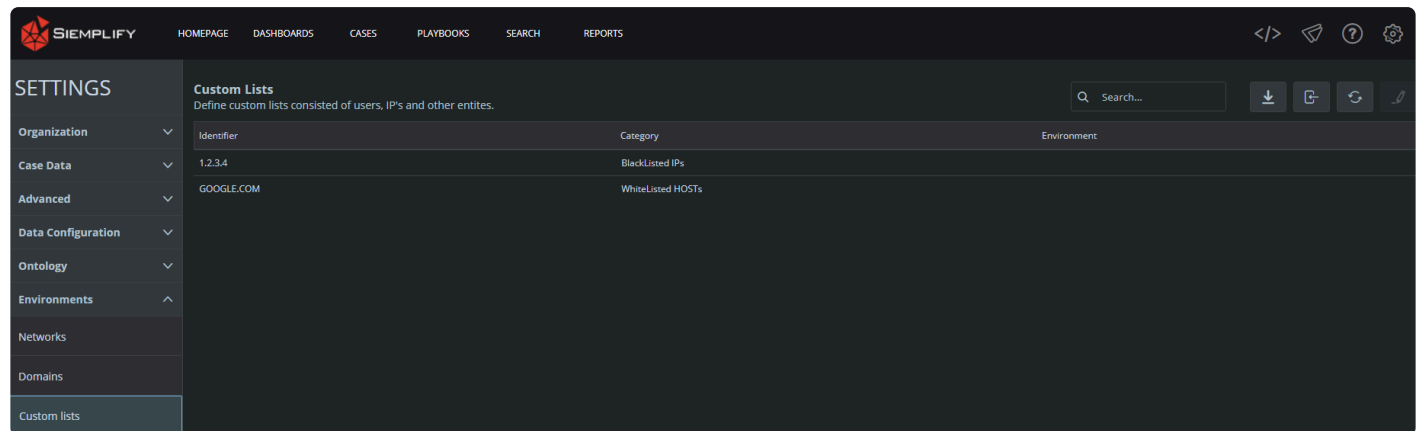
Sample Code

```
from SimplifyAction import SimplifyAction
```

```
siemplify = SiemplifyAction()
result = siemplify.add_alert_entities_to_custom_list("WhiteListed HOSTs")
```

Result Behavior

Adds the "Whitelisted HOSTs" category.



Result Value

```
[<SiemplifyDataModel.CustomList object at 0x0000000003476E10>, <SiemplifyDataModel.CustomList object at 0x0000000003476B00>]
```

4.2.3.5. add_tag

✿ Related Concepts: [Case Manipulation](#)

This function adds a single tag to the current Simplify case. A tag can then be later used to filter the case queue, search or dashboard widgets.

```
add_tag(tag)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
tag	string	Any string to be used as a tag	Length should be shorter than 250 characters	Yes
case_id	string	Takes by default the ID of the context case		No
alert_identifier	string	Takes by default the ID of the context alert		No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
tag_to_be_added = "MaliciousMail"
simplify.add_tag(tag=tag_to_be_added)
```

Result Behavior

```
"MaliciousMail" tag is added to the current case.
```

Result Value

None

NOTE: It is advised not to create tags that are too specific, as tags are used in the system to help search and filter cases. So, try to avoid using tags with entity identifiers or any other unique strings.

NOTE 2: Tags created either manually or by an action will not count for playbook trigger “By Tag”. For that, please refer to the “Case Tag” table in the Siemplify settings.

4.2.3.6. any_alert_entities_in_custom_list

This function gets a category name (From CustomLists in the Simplify settings) and returns True (Boolean) if any of the entities in the scope is in that category (an entity is considered in the category if its identifier is listed with this category in the Simplify settings on the CustomLists table).

NOTE: The Environment is added implicitly from the alert's environment!

```
result = simplify.any_entity_in_custom_list("BlackListed IPs")
```

Parameters:

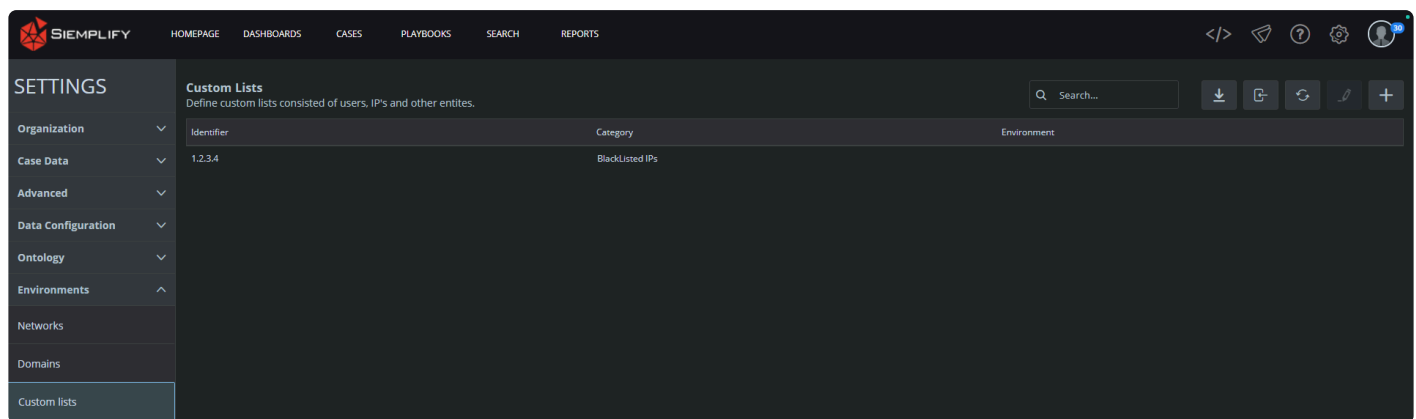
Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	"BlackListed IPs"	the custom list category	Yes

Return Type

Boolean

Example:

Input: Explicitly, category_name. Implicitly, entities via scope.



In the example above, if the IP “1.2.3.4” is part of the action’s scope, the following code will return True (Boolean):

Sample Code 1

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify.any_entity_in_custom_list("BlackListed IPs")
```

Sample Code 2

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.any_entity_in_custom_list("Executive IPs")
```

Result Behavior

Sampe Code 1 `_result_` is True. However, Sample Code 2 result is False.

Result Value

True/False

4.2.3.7. assign_case

✿ Related Concepts: [Case Manipulation](#)

This function assigns the current case to the user. This function requires the user to whom case is going to be assigned.

```
assign_case(assigned_user)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
user	string	Username or role to which case will be assigned. e.g. Admin/@tier1		Yes
case_id	string	Case Identifier e.g. 30123		No
alert_identifier		This value is fetched during the run time of the action.		No

Return Type

```
NoneType
```

Example

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
assigned_user= "Admin"
simplify.assign_case(assigned_user)
```

Result Behavior

The case gets assigned to the Admin user.

Result Result

None

4.2.3.8. attach_workflow_to_case

This function attaches workflow to case.

```
attach_workflow_to_case(workflow_name, cyber_case_id, indicator_identifier)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
workflow_name	string	workflow name	N/A	Yes
cyber_case_id	string	case identifier	If provided, the cyber_case_id is the unique case identifier	No
indicator_identifier	string	alert_identifier	If provided, the indicator_identifier is the unique alert identifier	No

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.attach_workflow_to_case(workflow_name, cyber_case_id, indicator_identifier)
```

Result Behavior

Attaches the given workflow to the case for the given indicator identifier

Result Value

```
None
```

4.2.3.9. change_case_priority

✿ Related Concepts: [Case Manipulation](#)

This function sets a case's priority to a specific value. Values for priority are integers and will be described below.

```
siemplify.change_case_priority(priority=-1, 40, 80 or 100)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
priority	int	One of the following: -1, 40, 60, 80 or 100	Priority represented by each number respectively is: Informative, Low, Medium, High and Critical	Yes
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
priority_to_change_to = 60
siemplify.change_case_priority(priority=priority_to_change_to )
```

Result Behavior

The case priority gets changed to "Medium".

Result Value

```
None
```

NOTE: *Case Priority* is a case's property! Be mindful when changing it from a playbook/alert's perspective.

4.2.3.10. change_case_stage

✿ Related Concepts: [Case Manipulation](#)

This function sets case's stage to a specific chosen stage. Stage's name must be one of the values specified in the Simplify settings table – *case stages*.

```
siimplify.change_case_stage(stage=stage_to_change_to)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
stage	string	Any string from the <i>case stages</i> table	Stage should match exactly the string that is defined in the <i>case stages</i> table	Yes
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiimplifyAction import SiimplifyAction
siimplify = SiimplifyAction()
stage_to_change_to = "Investigation"
siimplify.change_case_stage(stage=stage_to_change_to)
```

Result Behavior

The case state is changed to "investigation".

Result Value

```
None
```


NOTE: *Case Stage* is a case's property! Be mindful when changing it from a playbook/alert's perspective.

4.2.3.11. close_case

✿ Related Concepts: [Case Manipulation](#)

This function closes the current case. This is the same as manually closing the case. Function requires the reason for closure, a root cause and a comment.

```
siemply.close_case(reason=reason, root_cause=root_cause, comment=comment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
reason	string	One of three predefined strings available in the close popup (when done manually): "NotMalicious", "Malicious" and "Maintenance"	When creating an action, there is a parameter type named "Case close reason" that can be used. The user will have to choose from the three possible reasons and won't be able to input their own strings	Yes
root_cause	string	A string taken from the "Case close root cause" table in the settings	Similar to the previous parameter, there is an action parameter type called "Close case root cause" which forces the user to choose from values available in the relevant table	Yes
comment	string	Any string could be used here	Comment should describe the case, but is not restricted	Yes
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplyAction import SiemplyAction
```

```
siemplyfy = SiemplyfyAction()  
reason = "Maintenance"  
root_cause = "Employee Error"  
comment = "User accidentally activated a correlation before it was ready to be used and triggered this alert"  
siemplyfy.close_case(reason=reason, root_cause=root_cause, comment=comment)
```

Result Behavior

The case gets closed with the specified reason, root cause and comment.

Result Value

None

4.2.3.12. close_alert

✿ Related Concepts: [Case Manipulation](#)

This function closes the current alert. This is the same as manually closing the alert from the case overview. Function requires the reason for closure, a root cause and a comment, just like the close case alert.

Closing an alert in Simplify does the following:

1. Moves the current alert (the one we close) to a newly created case
2. Closes the new case (with only one alert)

```
siemplify.close_alert(reason=reason, root_cause=root_cause, comment=comment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
reason	string	One of three predefined strings available in the close popup (when done manually): "NotMalicious", "Malicious" and "Maintenance"	When creating an action, there is a parameter type named "Case close reason" that can be used. The user will have to choose from the three possible reasons and won't be able to input his own strings	Yes
root_cause	string	A string taken from the "Case close root cause" table in the settings	Similar to the previous parameter, there is an action parameter type called "Close case root cause" which forces the user to choose from values available in the relevant table	Yes
comment	string	Any string could be used here	Comment should describe the case, but is not restricted	Yes
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
reason = "Maintenance"
root_cause = "Employee Error"
comment = "User accidentally activated a correlation before it was ready to be used and triggered this alert"
siemplify.close_alert(reason=reason, root_cause=root_cause, comment=comment)
```

Result Behavior

the current alert is moved to new case and subsequently closed with the alert.

Result Value

None

4.2.3.13. create_case_insight

✿ Related Concepts: [Insights](#)

This function creates a case insight.

```
create_case_insight(triggered_by, title, content, entity_identifier, severity, insight_type, additional_data, additional_data_type, additional_data_title)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
triggered_by	string	Virustotal, XForce	The triggered by value is the name of the integration.	Yes
title	string	insight title	Enriched by Virustotal.	Yes
content	string	Insight Message	Insight message to display.	Yes
entity_identifier	string	1.1.1.1, simplify.co	entity identifier.	Yes
severity	int	0, 1 or 2	0=info, 1 = warning, 2 = error	Yes
insight_type	int	0 or 1	0 = general, 1 = Entity	Yes
additional_data	Dictionary	{"checked against": "VT", "malicious": "No"}	Additional data to be added to the case insight.	No
additional_data_type	string	dict	type of the additional_data like list, dict or string.	No
additional_data_title	String	VT Check	Suitable title for the additional_data	No

Return Type

```
Boolean
```

Example

Sample Code

```
from SimplifyAction import SimplifyAction
```

```
siemplify = SiemplifyAction()  
siemplify.create_case_insight(triggered_by, title, content, entity_identifier, severity, insight_type, additional_data, additional_data_type, additional_data_title)
```

Result Behavior

Creates the insight for case with defined data.
True if case insight is created otherwise False.

Result Value

True/False

4.2.3.14. extract_action_param

✿ Related Concepts: [Integration Configuration & Script Parameters](#)

Get the value of an action parameter. Each action has parameters that are filled when the action is configured (in playbook or as manual action). This method allows extracting the value of a selected parameter of the currently running action.

```
param_value= simplify.extract_action_param(  
    param_name,  
    default_value=None,  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
param_name	string	Any of the parameters names available for the action	The name of the parameter to fetch	Yes
default_value		Any desired value	The default value of the parameter. The given value will be returned if the parameter was not set (if <code>is_mandatory</code> is set to <code>False</code>). Defaults to <i>None</i> .	No
input_type		Any valid python type	The type of the parameter. The returned value will be cast to the selected input type. Defaults to <i>str</i> .	No
is_mandatory	boolean	True/False	Whether the parameter is mandatory. If set to <i>True</i> and the parameter was not filled, an exception will be raised. Default to <i>False</i> .	No
print_value	boolean	True/False	Whether to output the fetched value of the parameter to the logs. Default to <i>False</i> .	No

Return Type

```
As passed in input_type
```


Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
param_value= siemplify.extract_action_param(
    "Threshold",
    default_value=-1,
    input_type=int,
    is_mandatory=False,
    print_value=False)
```

Result Behavior

The value of the selected parameter will be returned, casted to selected type.

Result Value

20

4.2.3.15. get_alerts_ticket_ids_from_cases_closed_since_timestamp

✿ Related Concepts: [Case Manipulation](#)

This function retrieves alerts from cases that were closed since timestamp.

```
get_ticket_ids_for_alerts_dismissed_since_timestamp(timestamp_unix_ms, rule_generator)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
logs_collector	string	"logs collector"	N/A	Yes

Return Type

None

Example:

Sample Code

```
from SimplifyBase import SimplifyBase
simplify = SimplifyBase()
simplify.set_logs_collector(logs_collector)
```

Result Behavior

Sets the logs collector for logging.

Result Value

N/A

4.2.3.16. get_attachments

This function gets a list of custom list items from category and entities list. This function returns a list of custom list item objects.

```
result = simplify.get_attachments(caseid)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	234	Unique case id	No

Return Type

```
Dict
```

Example:

Input: Explicitly, case id.

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify.get_attachments(case_id="234")
```

Result Behavior

A list of dictionaries of attachments will be returned for the case id 234.

Result Value

```
[{u'is_favorite': False, u'description': u'test', u'type': u'.exe', u'id': 4, u'name': u'chrome_proxy'}]
```

4.2.3.17. get_case_comments

✿ Related Concepts: [Case Manipulation](#)

This function gets the comments from the provided case.

```
get_case_comments(case_id)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	12314	Unique case id	No

Return Type

```
List
```

Example

Sample code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
simplify.get_case_comments(case_id)
```

Result Behavior

All comments belonging to case will be fetched.

Result Value

```
[
    {
        u 'comment': u 'Test',
        u 'case_id': 10085,
        u 'is_favorite': False,
        u 'alert_identifier': None,
        u 'creator_user_id': u 'Admin',
        u 'type': 5,
```

```
    u 'id': 1,  
    u 'modification_time_unix_time_in_ms': 1563272078332L  
  }, {  
    u 'comment': u 'jhfksth',  
    u 'case_id': 10085,  
    u 'is_favorite': False,  
    u 'alert_identififier': None,  
    u 'creator_user_id': u 'Admin',  
    u 'type': 5,  
    u 'id': 2,  
    u 'modification_time_unix_time_in_ms': 1563272079941L  
  }, {  
    u 'comment': u 'kjfhshdm',  
    u 'case_id': 10085,  
    u 'is_favorite': False,  
    u 'alert_identififier': None,  
    u 'creator_user_id': u 'Admin',  
    u 'type': 5,  
    u 'id': 3,  
    u 'modification_time_unix_time_in_ms': 1563272080598L  
  }  
]
```

4.2.3.18. get_configuration

✿ Related Concepts: [Integration Configuration & Script Parameters](#)

This function retrieves the stored configurations of the integration for the current running action. The only mandatory parameter is the “Integration Provider”, which is essentially the integration’s name. This identifier is the same string used in the integration’s definition file. The output of the function is a dictionary with all the properties found in the store’s database that matches the integration’s provider name. For example, if you want to write an action for “Active Directory”, you need to use the following code:

```
conf = siemplify.get_configuration("ActiveDirectory")
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
provider	string	Any one of the integration names from the marketplace	Integration Provider name is case sensitive. An error will be thrown if the integration is not installed or the string does not exist	Yes
environment	string	Environment name from the settings	<i>Optional</i> If provided, the credentials will be taken from the relevant environment's configuration. If no environment is stated, the case's environment is used by default. If there is no configuration for that specific environment, the default configuration will be returned	No

Return Type

Dictionary

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()

conf = siemplify.get_configuration("ActiveDirectory")
```

```
username = conf["Username"]
```

Result Behavior

Fetches relevant active directory configuration and returns it as a dictionary.

Result Value

```
{"Server": "Server Address", "Username": "Username", "Password": "Password", "Domain": "Domain", "Custom Fields": "Custom Fields", "Use SSL": "true"}
```

4.2.3.19. get_similar_cases

✿ Related Concepts: [Case Manipulation](#)

This function returns a dictionary of similar cases based on entities, ports, rule generators and category outcome in the provided time frame.

```
result = simplify.get_similar_cases(consider_ports,
                                    consider_category_outcome,
                                    consider_rule_generator,
                                    consider_entity_identifiers,
                                    days_to_look_back, case_id=None, end_time_unix_ms=None)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	234	Unique case id	Yes
consider_ports	boolean	True/False	This will configure whether to use port filter or not.	Yes
consider_category_outcome	boolean	True/False	This will configure whether to consider category outcome of the events.	Yes
consider_rule_generator	boolean	True/False	This will configure whether to consider rule generator for the alerts.	Yes
consider_entity_identifiers	boolean	True/False	This will configure whether to consider entity identifiers for the alerts.	Yes
days_to_look_back	integer	365	This will configure number of days backwards to look for similar cases.	Yes
end_time_unix_ms	string	1564214708469	The provided unix time is in milliseconds.	No

Return Type

List

Example:

Input: Everything needs to be explicitly provided except `case_id` and `end_time_unix_ms` as they can be implicitly extracted from the current case.

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.get_similar_cases(consider_ports=True,
                                     consider_category_outcome=False,
                                     consider_rule_generator=False,
                                     consider_entity_identifiers=False,
                                     days_to_look_back=30, case_id="234", end_time_unix_ms=None)
one)
```

Result Behavior

A list of case id similar to the case 234 will be returned.

Result Value

```
[4, 231]
```

4.2.3.20. load_case_data

This function loads the case data.

```
result = siemplify.load_case_data()
```

Parameters:

No parameters are required.

Return Type

NoneType

Example:

Input: Implicitly, case via current case.

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify.load_case_data()
```

Result Behavior

The case data gets loaded.

Result Value

None

4.2.3.21. mark_case_as_important

✿ Related Concepts: [Case Manipulation](#)

This function marks a case as important. The importance mark can be either filtered in the search window or in the case queue. In addition, it is visible in the case queue without clicking on the case itself.

```
siemplify.mark_case_as_important()
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	12345	Unique case identifier.	No
alert_identifier	string	12345	Unique alert identifier.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.mark_case_as_important()
```

Result Behavior

```
The current case is marked as important.
```

Result Value

```
None
```

4.2.3.22. raise_incident

✿ Related Concepts: [Case Manipulation](#)

This function raises the current alert as incident.

```
raise_incident(case_id, alert_identifier)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
case_id	string	12345	Unique case identifier	No
alert_identifier	string	123123	Unique alert identifier	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.raise_incident(case_id, alert_identifier)
```

Result Behavior

```
The case raised to Incident status.
```

Result Value

```
None
```

4.2.3.23. remove_alert_entities_from_custom_list

This function gets a category name (From CustomLists in the Simplify settings) and returns a list of objects of type *CustomList* for any of the entities in the scope that were removed from the chosen category. (Refer to the *SimplifyDataModel* for more info)

NOTE: The Environment is added implicitly from the alert's environment!

```
result = simplify.remove_alert_entities_from_custom_list("WhiteListed HOSTs")
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	"WhiteListed HOSTs"	the custom list category	Yes

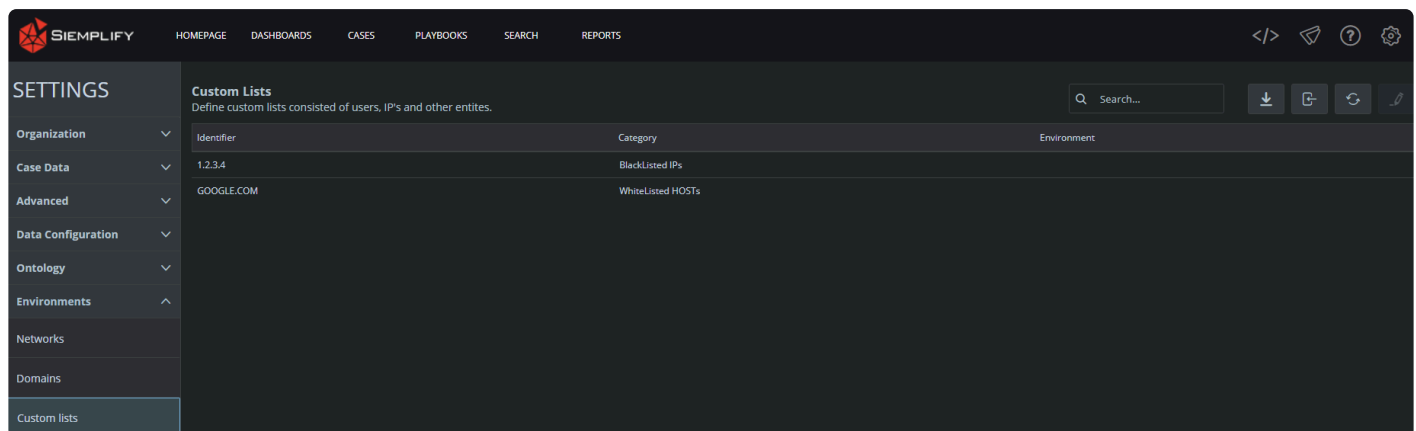
Return Type

```
List
```

Example:

Input: Explicitly, category_name. Implicitly, entities via scope.

Let's assume this is the state of the CustomList table prior to the function call, and let's assume the scope of the action has a single entity, "GOOGLE.COM"



Running `remove_alert_entities_from_custom_list` will result in a list of "CustomList" objects and a configuration change in the settings. Running the following code we get:

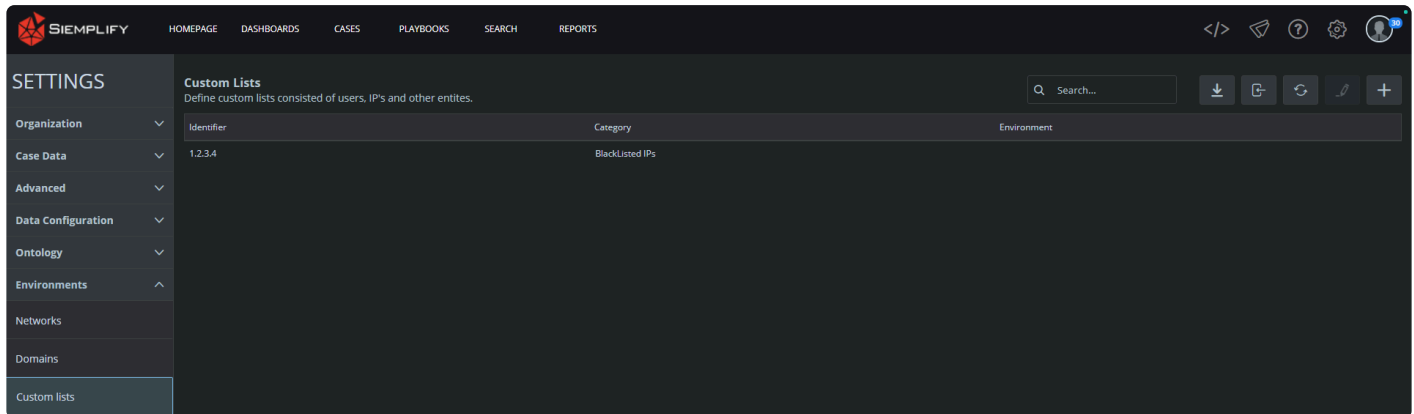
Sample Code

```
from SimplifyAction import SimplifyAction
```

```
siemplify = SiemplifyAction()
result = siemplify.remove_alert_entities_from_custom_list("WhiteListed HOSTs")
```

Result Behavior

The "WhiteListed HOSTS" is removed.



Result Value

```
[<SiemplifyDataModel.CustomList object at 0x0000000003476E10>, <SiemplifyDataModel.CustomList object at 0x0000000003476B00>]
```

4.2.3.24. set_logs_collector

This function retrieves alerts from cases that were closed since timestamp.

```
set_logs_collector(logs_collector)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
timestamp_unix_ms	long	1550409785000L	N/A	Yes
rule_generator	string	Phishing email detector	N/A	Yes

Return Type

```
List
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.get_alerts_ticket_ids_from_cases_closed_since_timestamp(timestamp_unix_ms=1550409785000L, rule_generator="Phishing email detector")
```

Result Behavior

The list of alerts from the cases that were closed since the timestamp are returned .

Result Value

```
[u'5792a6d6-0abd-40bc-a00a-2bffd7e4f122']
```

4.2.3.25. update_alerts_additional_data

This function updates the alerts with additional data from a playbook.

```
update_alerts_additional_data(alerts_additional_data, case_id)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
alerts_additional_data	dict	{"additionalData": "Value"}		Yes
case_id	string	12314	Unique case id. (Case Id is read dynamically while running the action)	No

Return Type

```
NoneType
```

Example

Sample code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
additional_data = {"testKey": "testValue"}
simplify.update_alerts_additional_data(alerts_additional_data=additional_data, case_id=caseid)
```

Result Behavior

```
updates the alert with additional data i.e. testKey:testValue.
```

Result Value

```
None
```


4.2.3.26. _get_case

This function returns the case object in dictionary format.

```
result = simplify._get_case()
```

Parameters:

No parameters are required.

Return Type

Dict

Example:

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify._get_case()
```

Result Behavior

Case data in the dictionary format is returned.

Result Value

```
{u'creation_time': 1564202116444L, u'alert_count': 1, u'assigned_user': u'@Tier 1', u'has_suspicious_entity': False, u'environment': u'', u'high_risk_products': None, u'has_workflow': False, u'title': u'IPS_Product', u'is_touched': False, u'is_merged': False, u'priority': -1, u'additional_properties': {}, u'sla_expiration_unix_time': None, u'status': 1, u'description': None, u'modification_time': 1564202116948L, u'is_incident': False, u'cyber_alerts': [...], u'is_important': False, u'stage': u'Triage', u'is_locked': False, u'identifier': u'234'}
```

4.2.3.27. _load_current_alert

This function loads the alerts from the case.

```
result = siemplify._load_current_alert()
```

Parameters:

No parameters are required.

Return Type

List

Example:

Input: Implicitly, alerts via scope.

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
result = siemplify._load_current_alert()
```

Result Behavior

An alert will be returned if it is present in the case.

Result Value

A list of alerts that are present in the current case.

4.2.3.28. _load_target_entities

This function adds the entity from alerts to the target entities list.

```
result = simplify._load_target_entities()
```

Parameters:

No parameters are required.

Return Type

None

Example:

Input: Implicitly, entities via scope.

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify._load_target_entities()
```

Result Behavior

Entities that are on target_entities will be added to the list.

Result Value

target_entities list will be updated with new entity.

4.2.3.29. _get_custom_list_items

✿ Related Concepts: [Custom Lists](#)

This function gets a list of custom list items from category and entities list. This function returns a list of custom list item objects.

```
result = simplify._get_custom_list_items(category_name, entities)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
category_name	string	"BlackListed IPs"	the custom list category	Yes
entities	list	["GOOGLE.COM", "8.8.8.8"]	a list of entities	Yes

Return Type

```
List
```

Example:

Input: Explicitly, category_name. Implicitly, entities via scope.

Sample Code

```
from SimplifyAction import SimplifyAction
simplify = SimplifyAction()
result = simplify._get_custom_list_items("Blacklisted IP", entities)
```

Result Behavior

```
A list of entities in the blacklisted IP category will be returned.
```

Result Value

```
List of entities in blacklisted IP category.
```

4.2.4. **SiemplifyConnectorExecution** **(SiemplifyConnectors.py)**

The SiemplifyConnectorExecution object inherits its properties from the Siemplify object, which inherits its properties from the SiemplifyBase object.

SiemplifyBase = Grandfather

Siemplify = Father

SiemplifyConnectorExecution = Child

4.2.4.1. is_overflowed_alert

This function checks whether a given alert will be overflowed during the case ingestion in Simplify system. Simplify has a builtin overflow prevention mechanism, based on multiple parameters, i.e: alert identifier, ingestion time, alert name and etc. An overflowed alert will not be ingested to the Simplify system, but marked as an overflow alert. This function allows to determine whether a given alert with certain parameters will be marked as an overflow during ingestion process or not.

```
is_overflowed_alert(environment, alert_identifier, ingestion_time=SimplifyUtil.s.unix_now(), original_file_path=None, original_file_content=None, alert_name=None, product=None, source_ip=None, source_host=None, destination_ip=None, destination_host=None)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
environment	string	Simplify, Apple	The environment of the alert. Environment can be created in the Simplify settings	Yes
alert_identifier	string	123123	Unique alert identifier	Yes
ingestion_time	long	Current Unix Time	If not provided, defaults to current time (UNIX time format).	Yes
original_file_path	string	Path to the file containing the alert's original raw data		No
original_file_content	string	Content of the file containing the alert's original raw data		No
alert_name	string	The name of the alert		No
product	string	McAfee ESM, QRadar	The product name for the device that generated the alert	No
source_ip	string	10.0.0.13, 192.168.0.13	Source IP address associated with the alert	No
source_host	string	source@company.local, source.company.local	The source host address associated with the alert	No
destination_ip	string	10.0.0.31, 192.168.0.31	Destination IP address associated with the alert	No

destination_host	string	remote.company.local	Destination host address associated with the alert	No
-------------------------	--------	----------------------	--	----

Return Type

Boolean

Example

Sample code

```
from SiemplifyConnectors import SiemplifyConnectorExecution
siemplify = SiemplifyConnectorExecution()
siemplify.is_overflowed_alert(environment, alert_identifier, ingestion_time=SiemplifyUtils.unix_now(), original_file_path, original_file_content, alert_name, product, source_ip, source_host, destination_ip, destination_host)
```

Result Behavior

True if the alert will be overflowed during ingestion process, otherwise False.

Result Value

True/False

4.2.4.2. return_package

This function allows to inject CaseInfo objects to Siemplify as cases. The function converts the CaseInfo objects to matching Siemplify case files, who are being transferred to the Siemplify Data Processing Pipeline Engine. The given CaseInfo objects will be converted to matching Siemplify case files and will be processed and injected to the Siemplify system.

```
return_package(cases, output_variables, log_items)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
cases	cases	list of cases [CaseInfo]	Cases ready to be ingested to siemplify system.	Yes
output_variables	dict		Deprecated	No
log_items	list		Deprecated	No

Return Value

None

Example

Sample code

```
from SiemplifyConnectors import SiemplifyConnectorExecution  
siemplify = SiemplifyConnectorExecution()  
siemplify.return_package(cases, output_variables, log_items)
```

Result Value

None

4.2.4.3. return_test_result

Deprecated.

4.2.4.4. extract_connector_param

✿ Related Concepts: [Integration Configuration & Script Parameters](#)

Get the value of a connector parameter. Each connector has parameters that are filled when the it's configured. This method allows to extract the value of a selected parameter of the currently running connector.

```
param_value= simplify.extract_connector_param(  
    param_name,  
    default_value=None,  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
param_name	string	some_param_value	The name of the parameter to fetch	Yes
default_value			The default value of the parameter. The given value will be returned if the parameter was not set (if is_mandatory is set to False). Defaults to <i>None</i> .	No
input_type			The type of the parameter. The returned value will be cast to the selected input type. Defaults to <i>str</i> .	No
is_mandatory	boolean	True/False	Whether the parameter is mandatory. If set to <i>True</i> and the parameter was not filled, an exception will be raised. Default to <i>False</i> .	No
print_value	boolean	True/False	Whether to output the fetched value of the parameter to the logs. Default to <i>False</i> .	No

Return Type

```
As passed in input_type
```

Sample Code

```
from SimplifyConnectors import SimplifyConnectorExecution
```

```
siimplify = SiimplifyConnectorExecution()  
param_value= siimplify.extract_connectors_param(  
    "Logs Folder",  
    default_value="C:\\Siimplify_Server\\Scripting\\JobLogs",  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Result Behavior

The value of the selected parameter will be returned, casted to selected type.

Result Value

```
C:\\Siimplify_Server\\Scripting\\SampleJob\\Logs
```

4.2.5. SimplifyJob (SimplifyJob.py)

The SimplifyJob object inherits its properties from the Simplify object, which inherits its properties from the SimplifyBase object.

SimplifyBase = Grandfather

Simplify = Father

SimplifyJob = Child

4.2.5.1. get_configuration

This function retrieves the stored credentials for the *current* running integration to be used in the Job. The provider parameter is mandatory and is case sensitive.

```
get_configuration(provider, environment)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
provider	string	Any one of the integration names from the marketplace	Integration Provider name is case sensitive. An error will be thrown if the integration is not installed or the string does not exist	Yes
environment	string	Environment name from the settings	<i>Optional</i> If provided, the credentials will be taken from the relevant environment's configuration. If no environment is stated, the case's environment is used by default. If there is no configuration for that specific environment, the default configuration will be returned	No

Return Type

```
Dictionary
```

Example

Sample Code

```
from SiemplifyJob import SiemplifyJob
siemplify = SiemplifyJob()
siemplify.get_configuration(provider="VirusTotal", environment="")
```

Result Behavior

Dictionary with saved credentials for the integration from the marketplace will be returned.

Result Value

```
{
  u'AgentIdentifier': None,
  u'Api Key': u'c0c412#####4f85b22e707',
  u'Verify SSL': u'True',
  u'RunRemotely': u'False'
}
```

4.2.5.2. extract_job_param

✿ Related Concepts: [Integration Configuration & Script Parameters](#)

Get the value of an job parameter. Each job has parameters that are filled when the job is configured. This method allows to extract the value of a selected parameter of the currently running job.

```
param_value= simplify.extract_job_param(  
    param_name,  
    default_value=None,  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
param_name	string	some_param_value	The name of the parameter to fetch	Yes
default_value			The default value of the parameter. The given value will be returned if the parameter was not set (if is_mandatory is set to False). Defaults to <i>None</i> .	No
input_type			The type of the parameter. The returned value will be cast to the selected input type. Defaults to <i>str</i> .	No
is_mandatory	boolean	True/False	Whether the parameter is mandatory. If set to <i>True</i> and the parameter was not filled, an exception will be raised. Default to <i>False</i> .	No
print_value	boolean	True/False	Whether to output the fetched value of the parameter to the logs. Default to <i>False</i> .	No

Return Type

```
As passed in input_type
```

Sample Code

```
from SimplifyJob import SimplifyJob  
simplify = SimplifyJob()
```

```
param_value= simplify.extract_job_param(  
    "Logs Folder",  
    default_value="C:\\Simplify_Server\\Scrip  
ting\\JobLogs",  
    input_type=str,  
    is_mandatory=False,  
    print_value=False)
```

Result Behavior

The value of the selected parameter will be returned, casted to selected type.

Result Value

C:\\Simplify_Server\\Scripting\\SampleJob\\Logs

4.2.5.3. get_system_info

This function retrieves the system information such as case information, user information and so on. See the Result Value section for all information returned by the function.

```
get_system_info(start_time_unixtime_ms)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
start_time_unixtime_ms	string	1564532753123	The Unix timestamp in milliseconds.	Yes

Return Type

```
Dictionary
```

Example

Sample Code

```
from SiemplifyJob import SiemplifyJob
import json
siemplify = SiemplifyJob()
siemplify.script_name = "Retrieve System Information"
systemInfo = siemplify.get_system_info(start_time_unixtime_ms="1564532753123")
print json.dumps(systemInfo)
```

Result Behavior

Json with system information will be printed to standard output from the provided timestamp.

Result Value

```
{
    "sla_count": 0,
    "unique_users_last_month": 1,
    "manual_actions_used": 0,
    "reports_generated": 0,
```

```
"new_integrations": [],
"new_connectors": [],
"escalations_to_tier2_count": 0,
"workbooks_with_close_action": 0,
"last_upgrade_date": "2019-06-20T16:01:49",
"action_playbook_appearances": [],
"marketplace_version": "12.14",
"average_opened_cases_per_user": 0.0,
"important_cases_count": 0,
"playbooks_executed": 0,
"top_user_screen_resolutions": ["1920x1080"],
"playbooks_edited": 0,
"average_alerts_per_day": 0.0,
"average_closed_cases_per_day": 0.0,
"widgets_created": 0,
"visualization_accessed": 0,
"searches_executed": 0,
"incidents_invoked_count": 0,
"average_tasks_per_case": 0.0,
"environments_count": 2,
"custom_actions_created": 0,
"average_insights_per_case": 0.0,
"case_comments_added": 0,
"version_number": "5.2.22.0",
"dashboard_shows": 0,
"theme_usages": [],
"users_created": 0,
"top_user_browsers": ["Mozilla/5.0,(X11; Linux x86_64),AppleWebKit/537.36,(KHTML, like Gecko),Ubuntu,Chromium/75.0.3770.90,Chrome/75.0.3770.90,Safari/537.36"],
"report_templates_edited": 0,
"case_reports_generated": 0,
"unique_users_last_day": 1,
"average_users_per_day": 1.7894736842105263,
"widgets_edited": 0,
"average_cases_per_day": 0.0,
"custom_actions_edited": 0,
"playbooks_created": 0
}
```

4.2.6. ScriptResult (ScriptResult.py)

This class represents the return object an action passes back to Simplify. The returned object consists of: JSON, links, tables and more.

4.2.6.1. add_entity_json

This function adds json result with entity identifier as title.

```
siemplify.result.add_entity_json(entity_identifier, json_data)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
json_data	dict	JSON formatted data		Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity_identifier = "10.0.0.1"
json_data = {"title":"10.0.0.1", "Message":"This is the default gateway"}
siemplify.result.add_entity_json(entity_identifier, json_data)
```

Result Behavior

The provided Json data will be added to entity 10.0.0.1.

Result Value

```
None
```

4.2.6.2. add_result_json

This function adds json result to the case.

```
siemplify.result.add_result_json(json_data)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
json_data	dict	JSON formatted data		Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
json_data = {"title":"10.0.0.1", "Message":"This is the default gateway"}
siemplify.result.add_result_json(json_data)
```

Result Behavior

The provided Json data will be added to current case.

Result Value

```
None
```

4.2.6.3. add_entity_content

This function adds json result with entity identifier as title.

```
siemplify.result.add_entity_content(entity_identifier, content)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
content	string	Data related to the entity to add		Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity_identifier = "10.0.0.1"
content = {"title":"10.0.0.1", "Message":"This is the default gateway"}
siemplify.result.add_entity_content(entity_identifier, json_data)
```

Result Behavior

The provided content data will be added to entity 10.0.0.1.

Result Value

```
None
```

4.2.6.4. add_entity_table

This function adds data table with entity identifier as table title.

```
siimplify.result.add_entity_table(entity_identifier, data_table)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
data_table	list	CSV formatted list		Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiimplifyAction import SiimplifyAction
siimplify = SiimplifyAction()
entity_identifier = "10.0.0.1"
data_table = ["h1", "h2", "h3", "Entity Type", "Enrichment", "Original Identifier"]
siimplify.result.add_entity_table(entity_identifier, data_table)
```

Result Behavior

The provided csv data will be added to entity 10.0.0.1 with entity identifier as title.

Result Value

```
None
```

4.2.6.5. add_entity_attachment

This function adds json result with entity identifier as title.

```
siemplify.result.add_entity_attachment(entity_identifier, filename, file_contents, additional_data=None)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
filename	string	File name of the attachment		Yes
file_contents	base64	File contents in the base 64 format		Yes
additional data	string	Any relevant attachment data		No

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity_identifier = "10.0.0.1"
file_contents = <base64 value>
filename = "Image.png"
siemplify.result.add_entity_attachment(entity_identifier, filename, file_contents, additional_data=None)
```

Result Behavior

The provided file will be added as attachment to entity 10.0.0.1.

Result Value

None

4.2.6.6. add_entity_html_report

This function adds html data with entity identifier as title.

```
siemplify.result.add_entity_html_report(entity_identifier, report_name, report_contents)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
report_name	string	HTML report name		Yes
report_contents	HTML	HTML contents of the report		Yes

Return Type

NoneType

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity_identifier = "10.0.0.1"
report_name = "HTML Report"
report_contents = "<html><body><title>html content</title></body></html>"
siemplify.result.add_entity_html_report(entity_identifier, report_name, report_contents)
```

Result Behavior

The provided html contents will be added to entity 10.0.0.1.

Result Value

None

4.2.6.7. add_entity_link

This function adds json result with entity identifier as title.

```
siemplify.result.add_entity_link(entity_identifier, link)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	entity identifier values such as 1.1.1.1, google.com	N/A	Yes
link	string	Link to the websites such as https://siemplify.co		Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
entity_identifier = "10.0.0.1"
link = "https://siemplify.co/entity/10.0.0.1"
siemplify.result.add_entity_link(entity_identifier, link)
```

Result Behavior

The provided link will be added to entity 10.0.0.1.

Result Value

```
None
```

4.2.6.8. add_link



Related Concepts: [Action Results](#)

This function adds a web link to the selected entity.

```
siemplify.add_link(title, link)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
title	string	Siemplify Homepage	Title for the link	Yes
link	string	https://siemplify.co	Website Link.	Yes

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.result.add_link("Siemplify", "https://siemplify.co")
```

Result Behavior

The provided website link gets added as result.

Result Value

```
None
```

4.2.6.9. add_attachment

✿ Related Concepts: [Action Results](#)

This function adds a web link to the selected entity.

```
siemplify.result.add_attachment(title, filename, file_contents, additional_data)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
title	string	Potentially Malicious File	Suitable Title for the attachment	Yes
filename	string	chrome_proxy.exe	Suitable filename for the attached file.	Yes
file_contents	base64		Base64 formatted file content.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.result.add_attachment(title="Malicious File", filename="virus.ini", file_contents="Base64 content of virus.ini", additional_data=None)
```

Result Behavior

The attachment gets added as result.

Result Value

```
None
```

4.2.6.10. add_content

This function adds a web link to the selected entity.

```
siemplify.result.add_content(entity_identifier, content)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
entity_identifier	string	"1.2.3.4"	Unique entity identifier	Yes
content	dict	content="some content"	Content is added as the key-value pair to the output of *_get_entity_data(entity_identifier) function.	No

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.add_content(entity_identifier="1.2.3.4", content="New content")
```

Result Behavior

The content gets added as result for the selected entity. Entity can be implicitly selected from the scope.

Result Value

```
None
```

4.2.6.11. add_html

This function adds a web link to the selected entity.

```
siemplify.result.add_html(title, report_name, report_contents)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
title	string	HTML File	Suitable Title for the HTML file	Yes
report_name	string	report.html	HTML file report.	Yes
report_contents	string	HTML	HTML content of the file as string.	Yes

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
siemplify.result.add_html(title="New evidence", report_name="evidence.html", report_content="<html><body><title>Evidence</title><p>This is html report</p></body></html>")
```

Result Behavior

The html report gets added as result for the selected entity. Entity can be implicitly selected from the scope

Result Value

```
None
```

4.2.6.12. add_json

✿ Related Concepts: [Action Results](#)

This function adds json result with entity identifier as title.

```
siemplify.result.add_json(entity_identifier, json_data)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Is Mandatory
entity_identifier	string	Any string to be used as title (Does not have to be an entity identifier!)	Using the same title for more than one object will bundle the objects together in the action result view	Yes
json_data	string OR dict	This parameter can be either a string representing a JSON (json.dumps() can help achieve it, see example above) or a 'python' dictionary	Sending the dictionary or dumping it into a string has the same effect. It is recommended to always dump the dictionary into a string. A list can only be sent as a string.	Yes

Return Type

```
NoneType
```

Example

Sample code

```
siemplify = SiemplifyAction()
import json
data = {
    "title": "Product",
    "type": "object",
    "required": ["id", "name", "price"],
    "properties": {
        "id": {
            "type": "number",
            "description": "Product identifier"
        },
    },
}
```



```
    "stock": {
      "type": "object",
      "properties": {
        "warehouse": {
          "type": "number"
        },
        "retail": {
          "type": "number"
        }
      }
    }
  }
}

simplify.result.add_json('Title goes here', json.dumps(data))
```

Result Behavior

The provided json data gets added as result for the selected entity. Entity can be implicitly selected from the scope

Result Value

None

4.2.6.13. add_data_table

✿ Related Concepts: [Action Results](#)

This function adds json result with entity identifier as title.

```
siemplify.result.add_data_table(title, data_table)
```

Parameters

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
title	string	Open Ports for this entity	Suitable title for the table	Yes
data_table	list	csv_data = ["entity,open ports", "1.2.3.4,80"]	CSV formatted list of information	Yes

Return Type

```
NoneType
```

Example

Sample code

```
from SiemplifyAction import SiemplifyAction
siemplify = SiemplifyAction()
title = "open ports per entity"
csv_data = ["entity,open ports", "1.2.3.4,80"]
siemplify.result.add_data_table(title=title, data_table=csv_data)
```

Result Behavior

```
The provided table is added as result.
```

Result Value

```
None
```

4.2.7. SiemplifyLogger (SiemplifyLogger.py)

4.2.7.1. loadConfigFromFile

This function loads logger config file.

```
loadConfigFromFile(run_folder=run_folder, log_location=log_location)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
run_folder	string	".\VirusTotal"	The running folder path.	Yes
log_location	string	elastic search log location		Yes

Return Type

```
Dictionary
```

Example:

Sample Code

```
from SiemplifyLogger import SiemplifyLogger
sb = SiemplifyLogger()
run_folder = "D:\Siemplify\ElasticSearch"
log_location = "D:\Siemplify\ElasticSearch\elastic.log"
sb.loadConfigFromFile(run_folder=run_folder, log_location=log_location)
```

Result Behavior

```
Configuration is loaded from the provided run folder and log location.
```

Result Value

```
{}
```

4.2.7.2. exception

This function logs a message with **ERROR** level in the Siemplify logs (both in log files and in Elasticsearch). In addition the traceback of the last exception will be written to the logs as well.

```
exception(message)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
message	string (python 3.7) / unicode (python 2.7)		The message to log. If an exception is caught, the exception object itself can be passed as well.	Yes

In addition to the message param, additional **custom parameters** can be passed. The added parameters will be displayed in the matching Elasticsearch document, under the **args** key and can be used to log additional information regarding the log entry, i.e: the line number, the alert id, etc.

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyLogger import SiemplifyLogger
sb = SiemplifyLogger()
try:
    result = 1/0
except Exception as e:
    sb.error("Division by Zero", alert_id=123)
```

Result Behavior

```
"Division by Zero" will be logged with level ERROR in the log file along with the exception traceback and a matching document will be created in Elasticsearch with args.alert_id field with value of 123.
```

Result Value

None

4.2.7.3. error

This function logs a message with **ERROR** level in the Siimplify logs (both in log files and in Elasticsearch).

```
error(message)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
message	string (python 3.7) / unicode (python 2.7)		The message to log	Yes

In addition to the message param, additional **custom parameters** can be passed. The added parameters will be displayed in the matching Elasticsearch document, under the **args** key and can be used to log additional information regarding the log entry, i.e: the line number, the alert id, etc.

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiimplifyLogger import SiimplifyLogger
sb = SiimplifyLogger()
message = "Error Message"
sb.error(message, alert_id=123)
```

Result Behavior

```
"Error Message" will be logged with level ERROR in the log file and a matching document will be created in Elasticsearch with args.alert_id field with value of 123.
```

Result Value

```
None
```

4.2.7.4. warn

This function logs a message with **WARN** level in the Simplify logs (both in log files and in Elasticsearch).

```
warn(message)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
message	string (python 3.7) / unicode (python 2.7)		The message to log	Yes

In addition to the message param, additional **custom parameters** can be passed. The added parameters will be displayed in the matching Elasticsearch document, under the **args** key and can be used to log additional information regarding the log entry, i.e: the line number, the alert id, etc.

Return Type

```
NoneType
```

Example:

Sample Code

```
from SimplifyLogger import SimplifyLogger
sb = SimplifyLogger()
message = "Warning Message"
sb.warn(message, alert_id=123)
```

Result Behavior

"Warning Message" will be logged with level WARN in the log file and a matching document will be created in Elasticsearch with args.alert_id field with value of 123.

Result Value

```
None
```


4.2.7.5. info

This function logs a message with **INFO** level in the Siemplify logs (both in log files and in Elasticsearch).

```
info(message)
```

Parameters:

Param Name	Param Type	Possible Values	Comments	Mandatory Parameter
message	string (python 3.7) / unicode (python 2.7)		The message to log	Yes

In addition to the message param, additional **custom parameters** can be passed. The added parameters will be displayed in the matching Elasticsearch document, under the **args** key and can be used to log additional information regarding the log entry, i.e: the line number, the alert id, etc.

Return Type

```
NoneType
```

Example:

Sample Code

```
from SiemplifyLogger import SiemplifyLogger
sb = SiemplifyLogger()
message = "Informational Message"
sb.info(message, alert_id=123)
```

Result Behavior

```
"Informational Message" will logged with level INFO in the log file and a matching document will be created in Elasticsearch with args.alert_id field with value of 123.
```

Result Value

```
None
```