

Action Editor Documentation

July 17, 2024

Exabeam

1051 E. Hillsdale Blvd, 4th Floor
Foster City, CA 94404

650.209.8599

Have feedback on this guide? We'd love to hear from you!
Email us at docs@exabeam.com

Disclaimer: Ensure that you are viewing the most up-to-date version of this guide by visiting the Exabeam Documentation Portal.

Copyright

All content in this document, including text, graphics, logos, icons, images, and video clips, is the exclusive property of Exabeam or its content suppliers and is protected by U.S. and international copyright laws. The compilation (meaning the collection, arrangement, and assembly) of all content in this document is the exclusive property of Exabeam and is also protected by U.S. and international copyright laws. The content in this document may be used as a resource. Any other use, including the reproduction, modification, distribution, transmission, republication, display, or performance, of the content in this document is strictly prohibited.

Copyright ©2024 Exabeam, Inc. All Rights Reserved.

Trademarks

Exabeam, the Exabeam logo, Threat Hunter, Smarter SIEM, Smart Timelines and Security Management Platform are service marks, trademarks or registered marks of Exabeam, Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. The marks and logos displayed in this document may not be used without the prior written consent of Exabeam or their respective owners.

Patents

Exabeam owns, and reserves all rights for, patents for Exabeam products and services, which may be protected under registered patents as well as patents pending.

Other Policies

For information regarding how Exabeam treats personally identifiable information, please review the Exabeam privacy policy at www.exabeam.com/privacy.

Feedback

Have feedback on this guide? We'd love to hear from you! Email us at docs@exabeam.com.

Disclaimer: Please ensure you are viewing the most up-to-date version of this guide by visiting the [Exabeam Documentation Portal](#).

Table of Contents

Action Editor	4
Customize an Out-of-the-Box Service Using Action Editor	5
Create a Custom Service Using Action Editor	6
Edit a Custom Service in Action Editor	7
Delete a Custom Service in Action Editor	8
Edit a Custom Service Configuration Document	9
Edit a Configuration Document in the Service	9
Edit <code>connector.py</code>	9
Edit <code>__init__.py</code>	10
Add Dependencies to <code>requirements.txt</code>	10
Create a Custom Action Using Action Editor	11
1. Add Basic Information about the Action	11
2. (Optional) Configure Action Inputs	11
3. Configure Action Outputs	12
4. Customize the Workbench Output	12
Select the Workbench Output Type	12
Configure a Table Workbench Output	13
Configure a Map Workbench Output	13
5. Edit the Action Module	13
Send Data to a Table Workbench Output	14
Send Data to a Map Workbench Output	15
Create Error Messages and Exceptions	15
6. Edit the Service Configuration Documents	15
7. Download and Upload Your Service to Incident Responder	16
Edit a Custom Action in Action Editor	17
Delete a Custom Action in Action Editor	18

Action Editor

Create your own Incident Responder service and actions using Action Editor on Exabeam SOC Platform.

Action Editor is an application available for free on Exabeam SOC Platform to all Incident Responder users. It guides you through the process to customize an out-of-the-box service and actions or create your own custom service and actions from scratch.


You might consider creating a custom service or action if you can't build a specific workflow in a Incident Responder playbook using just out-of-the-box actions. For example, when you create a playbook, you can use an output of one node as an input for another node only if it takes in data of the same type. With Action Editor, you can change an action's inputs and outputs, or add more inputs and outputs, to ensure playbook nodes map together. You can also add more actions to an out-of-the-box service; create a service Incident Responder doesn't currently support so you get started quickly without waiting for Exabeam to develop it; or build other services and actions to fit your team's specific needs.

After you create a service or action in Action Editor, download a ZIP file, then upload it to Incident Responder. Your custom service and action is available for you to run manually or use to create your perfect playbook. To upload a service you created in Action Editor, you must have Incident Responder i53 or later.

Customize an Out-of-the-Box Service Using Action Editor

If an out-of-the-box service doesn't do what you need it to, clone the service so you can tweak and tailor it to your needs, without creating a service from scratch. You can only clone a service once.


Consider customizing an out-of-the-box service to modify an action's input or output fields to better suit your needs or add more actions to an existing service.

1. Select the service. To search for a specific service, enter the service name in the search bar.
2. Click clone .
3. Click **Clone**. Another service of the same name appears with a **Custom** label. All related configuration documents and actions are also cloned.

When you [upload](#) this service to Incident Responder, all related actions and playbooks will start using this custom service instead of the out-of-the-box service. To upload the service, you must have version i53 or above.



Create a Custom Service Using Action Editor

To quickly get started with a service Incident Responder doesn't currently support or to build a unique service specific to your needs, create your own service from scratch.

1. Click **+ Add a Service**.
2. Enter basic information about the service:
 - **Service name** – Enter a name that helps you quickly and easily identify the service; for example, Microsoft Outlook. It appears when you configure an action node in a playbook and select a service.
 - **Product name** – Enter the product name; for example, Office 365. It appears in the service settings.
 - **Description** – Describe the service, what it does, and what it's used for.
3. A connection parameter field defines information you enter to connect to the service; for example, username, password, domain, or TCP port. These fields appear when you add a service in Incident Responder service settings. To create a connection parameter field, click **+ Add a field**, then enter information about the field:
 - **Name** – Enter a name for the field.
 - **Display name** – Enter the name for the field that is displayed in Incident Responder.
 - **Description** – Describe the field.
 - **Data type** – Select **URL**, **string**, **password**, **long**, **boolean**, **large text**, or **picklist**.
If you select **picklist**, add the possible list values you select from: click edit , click **+Add picklist option**, type the name of the list value, then press the **enter** or **return** key.
If the field is required, select the **Required** checkbox.
4. Click **Save**.
Action Editor automatically creates three empty documents for the service: `__init__.py`, `connector.py`, and `requirements.txt`. Populate them [directly in the service](#) or when you [create](#) a custom action.


Edit a Custom Service in Action Editor

Rename the service or product, change the service description, or edit the configuration fields.

1. Select the service, then click edit .
2. Change the **Service Name**, **Product Name**, **Description**, or **Configuration Fields**.
If you cloned an out-of-the-box service, you can't edit the **Product Name** or **Version** because it's cloned from the original out-of-the-box service.
3. Click **Save**.
4. To download the service, select the download  icon, then download the ZIP file.
5. In Incident Responder, [upload](#) the service. You must have version i53 or later.

Delete a Custom Service in Action Editor

Delete a custom service [you created](#).

1. Select the service.
2. Click the trash .
3. Click **Delete**. If you don't want to use this service in Incident Responder, ensure that you also [delete](#) it there.

Edit a Custom Service Configuration Document

To build the logic behind a custom service and how a custom action communicates with it, edit the `__init__.py`, `connector.py`, and `requirements.txt` files.

Each custom service has three configuration documents: `__init__.py`, `connector.py`, and `requirements.txt`. These documents determine how all actions for a service communicate with it and are required for Incident Responder to process and upload the service.

When you first create a custom service, the documents are empty. You can populate a document in two ways: [directly in the service itself](#), and when you [create](#) or [edit](#) a custom action.

Edit a Configuration Document in the Service

1. Select a custom service.
2. Under **Documents**, select a document: `__init__.py`, `connector.py`, or `requirements.txt`.
3. Make changes to the document.
4. Click **SAVE**.

Edit `connector.py`

Build the logic the action uses to communicate — send and receive all requests — to the service.

The connector must:

- Define a class named after the service.
- Define the `__init__` method for this class so it assigns values to the service's configuration fields.

For example, if you created service configuration fields for `username` and `password`, the `__init__` method takes `username` and `password` as parameters and assigns them to objects:

```
def __init__(self, username: str=None, password: str=None) -> None:
    self.username = username
    self.password = password
```

- Define the `test_connection` method so if the configuration fields are valid, it returns `True`; if the configuration fields are invalid, it returns `False` or an exception.

For example, if you created configuration fields for `username` and `password`:

```
def authorize(self) -> bool:
    if not self._username or not self._password:
        return True

    res = self._session.post("auth/login", json={"username": self._username,
"password": self._password})
    if not res.ok:
        raise ExabeamConnectorException("Invalid credentials")
```

```
def test_connection(self) -> bool:
    return self.authorize()
```

Add any other logic necessary for actions to communicate to the service.

Edit `__init__.py`

Import all actions classes, the connector, and the test connection method, so it's contained in one location.

- Each time you create a new action for a service, you must update `__init__.py` so it imports the action class:

```
from .modulename import actionclass
```

For example:

```
from .get_reputation import GetFileReputation, GetIPReputation,
GetURLReputation
```

You must import all action classes. If you don't import it, you can't use it in Incident Responder .

- The connector contains methods for all actions and makes all API calls. Import the `connector` class under the alias, `Connector`:

```
from .connector import yourservice as Connector
```

- To call `test_connection`, a method in the `connector` class used to test the service, import the `TestServiceConnection` class under an alias. The alias should be *TestService* followed by the service name, without spaces:

```
from soar.library.common.test_connection import TestServiceConnection as
TestServiceYourService
```

For example:

```
from soar.library.common.test_connection import TestServiceConnection as
TestServiceCode42
```

If your service does not support connection testing, import the `TestServiceConnectionNotSupported` class instead of `TestServiceConnection` from the same package, under the same alias:

```
from soar.library.common.test_connection import
TestServiceConnectionNotSupported as TestServiceCode42
```

Add Dependencies to `requirements.txt`

In `requirements.txt`, list all external dependencies and libraries you must install to use `connector.py` and `__init__.py`.

Create a Custom Action Using Action Editor

After you [create](#) a custom service from scratch or [clone](#) an out-of-the-box service, create a custom action.

1. Add Basic Information about the Action

Name and describe the action to help you identify it when you configure an action node and manually run an action. The action name also appears in the service settings under service details.

Selecting an action type automatically populates the description and inputs for the service. If you don't find an action type that best describes your custom action, create a new action type.

1. Select a custom service, then next to **Actions**, click **+**.
2. Enter basic information about the action:
 - **Action name** – Enter a name for the action. This appears when you select a service to configure a playbook action node.
Since the action name is used throughout the action's back end, you must carefully choose the action name and review it before you continue. If you return to this step to rename the action, you lose all your work and must reconfigure the action from scratch.
 - **Action type** – An action type defines an action's basic metadata, inputs, and outputs. To automatically populate the action's description, inputs, and outputs, select an action type that best describes your action from the list.
If you don't see an action type that describes your action, create your own action type. In the **Select action type** field, start typing, then click **Add [action type]**. If you create your own action type, you must enter a description and define all inputs and outputs.
 - **Description** – Describe the action, what it does, and what it's used for. This description appears when you manually run an action in the workbench.
3. Review the action name, then click **Next**.

2. (Optional) Configure Action Inputs

An input is a parameter passed to the action and used to produce an output. You enter a value for the input when you manually run an action or configure a playbook action node.

You don't have to configure inputs if the action can get the information it needs from within the service host, using the information you provided when you configured the service. For example, the out-of-the-box *List Context Tables* action doesn't require any inputs because the Advanced Analytics API can retrieve a list of context tables without you providing additional information.

1. Click **+ Add an Input**.
2. Fill in information about the input:
 - **Name** – Enter a name for the input. You refer to this name in Python code you write later.
 - **Display name** – Enter a name that appears when you configure an Incident Responder playbook action node and select an input.

- (Optional) **Description** – Describe the input.
 - **Data type** – Select the intended data type of the input value: **URL, String, Password, Long, Boolean, Large_text, Hash, Picklist, File Artifacts, or File Entity**.
3. If you can enter or select multiple input values, select the **ALLOW MULTIPLE** checkbox. If you can only enter or select one input value, don't select the checkbox.
 4. If you must configure this input to run the action, select the **REQUIRED** checkbox. If you don't need to configure this input to run the action, don't select the checkbox.
 5. If you can type in text as an input value, select the **ALLOW USER INPUT** checkbox. If you can only select a pre-defined input value, don't select the checkbox.

3. Configure Action Outputs

An output is a parameter that the action returns after it processes and uses the input parameter.

In a playbook, the output of an action node is used as an input for the next node. The final output of a playbook, and the output for an action you manually run, appear in an incident's workbench.

1. Click **+ Add an Output**.
2. Fill in information about the output:
 - **Name** – Enter a name for the output. You refer to this name in Python code you write later.
 - **Display name** – Enter a name that appears when you configure an Incident Responder playbook action node and use the output of the previous node as the input.
 - (Optional) **Description** – Describe the output.
 - **Data type** – Select the intended data type of the output value: **URL, String, Password, Long, Boolean, Large_text, Hash, Number, or Timestamp**.
 - **Data path** – After you execute the action, the output is stored in a JSON file. Enter the file path for the JSON file. This file path is passed to the next playbook node as an input.
3. If the action can output a list of multiple values, select the **ALLOW MULTIPLE** checkbox.
4. Click **Next**.

4. Customize the Workbench Output

After you manually run an action or playbook, the outputs appear in an incident's workbench. Configure how the outputs of your custom action appear in the workbench.

Select the Workbench Output Type

Select the type of output that appears in the workbench after you run an action.

- If you don't want any output to appear in the workbench, select **No Card**, then click **Next**.
- To list action outputs in a table, select **Table**.
- To display action outputs on a world map, select **Map**. For example, the out-of-the-box *Geolocate IP* action produces a map-type workbench output.

Configure a Table Workbench Output

1. Under **Header Title**, enter a title for the output header.
2. Under **Header Color**, select a color for the output header. To select from a color picker or enter a RGBA value, click the color swatch. To enter a hex code, start typing after #.
3. Under **Empty Result Message**, enter a helpful message that displays in Incident Responder when the action can't get any data.
4. Configure table columns:
 - **Column label** – Enter a header name for the column.
 - **Data type** – Select **String**. Indicates the expected data type for the values in the column.
 - **Column width** – In the text field, enter a number in pixels (px) or as a percent (%). From the list, select the corresponding unit. If you create multiple columns, it's easiest to use the same units across all columns. If you enter a number in percent, ensure the numbers across all columns sum to 100 percent.

To add another column, click **+ Add a Column**.

5. Click **Next**.

Configure a Map Workbench Output

1. (Optional) Under **Header Title**, enter a title for the output header.
2. Under **Header Color**, select the color of the output header. To select from a color picker or enter a RGBA value, click the color swatch. To enter a hex code, start typing after #.
3. (Optional) Under **Empty Result Message**, enter a helpful message that displays in Incident Responder when the action can't get any data.
4. Click **Next**.

5. Edit the Action Module

Under **ACTION RELATED FILES**, in the module for the action, build all logic necessary for it to function, including how inputs are computed into outputs, what error messages are shown, and how action outputs are displayed in the workbench.

In any action module, you must:

- Import the `SoarAction` parent class:

```
from soar.library.common.soar_action import SoarAction
```

- Define a class that inherits from the `SoarAction` parent class, and name it after the action:

```
class youraction(SoarAction):
```

For example:

```
class GetGeolocationIpApi(SoarAction):
```

Ensure that you import this action class into `__init__.py`.

- Underneath the action class, create a method named `action`:

```
def action(self) -> bool:
```

Underneath this method, insert all action-related logic. When you run an action in Incident Responder, it starts reading from this point.

To set an action input:

```
actioninput = self.get_input_value_with_default('actioninput',  
'defaultinputvalue'  
)
```

If the action can't set the input, it returns the default input value. This parameter is optional.

- If you don't set a default input value, it returns `None`. For example:

```
ips = self.get_input_value_with_default('ips')
```

- If you set a default input value, it returns that value. For example:

```
ips = self.get_input_value_with_default('ips', '8.8.8.8')
```

If the action can't set the input, it returns `8.8.8.8`.

To set an action output and save it so it can be passed along and used as inputs in playbook nodes:

```
self.set_result({'actionoutput': 'outputvalue'})
```

Send Data to a Table Workbench Output

If you previously [configured](#) a table workbench output, specify the exact data that appears in the output.

To add a row to the table:

```
self.builder.add_row(datakey, [columnvalues])
```

For example:

```
self.builder.add_row(now, [host, status])
```

To indicate that there's no more data to add and display the workbench output:

```
self.add_display(self.builder.build())
```

Display the **Empty Result Message** in Incident Responder when the action can't get any data:

```
self-builder.add_empty_data(now)
```

If you didn't set an **Empty Result Message**, the workbench output displays *Action ran successfully but returned no results*.

Send Data to a Map Workbench Output

If you [previously configured](#) a map workbench output, specify the exact data that appears in the workbench output.

To set coordinates for and display an exact point on the map:

```
.set_coord(youroutputvalue, lat=latitude, lng=longitude,
markerContent=youroutputvalue)
```

`markerContent` is an optional parameter that uses a marker to identify a location on the map.

To indicate that there's no more data to add and display the workbench output:

```
self.add_display(self.builder.build())
```

Display the **Empty Result Message** in Incident Responder when the action can't get any data:

```
self.builder.add_empty_data(now)
```

If you didn't set an **Empty Result Message**, the workbench output displays *Action ran successfully but returned no results*.

Create Error Messages and Exceptions

If something goes wrong, it's helpful to provide error messages, exceptions, and other information about what happened so you can identify and debug the problem.

To import an Exabeam exception:

```
from soar.utility.custom_exceptions import ExabeamConnectorException
```

When you import the `SoarAction` class, you also import a logging package. Use it to print error messages or exceptions to a console or write them to a file:

```
self.logger.logginglevel('yourmessage')
```

For example:

```
self.logger.debug("No IP address found in input")
```

6. Edit the Service Configuration Documents

Under **GENERAL FILES**, [make](#) the necessary changes to `connector.py`, `__init__.py`, and `requirements.txt`. These documents determine how all actions for a service communicate with it and are required for Incident Responder to process and upload the service.

Ensure that you update `__init__.py` so it imports the action class you created in the action module:

```
from .modulename import actionclass
```


For example:

```
from .get_reputation import GetFileReputation, GetIPReputation, GetURLReputation
```

You must import all action classes. If you don't import it, you can't use it in Incident Responder .



7. Download and Upload Your Service to Incident Responder

After you create all custom actions and you're ready to use your custom service, download and upload it to Incident Responder .

1. In your custom service, click the download  icon. The service downloads as a ZIP file.
2. In Incident Responder, [upload](#) the service. You must have version i53 or later.



Edit a Custom Action in Action Editor

Rename, change inputs and outputs, customize the workbench, or modify any configuration files for a custom action you created.

1. Select the custom service, hover over the custom action, then click the More  menu.
2. Select **Edit Action**.
3. Edit the action name, type, or description; change inputs or outputs; customize the workbench; or modify the action module or [service configuration documents](#).
Since the action name is used throughout the action's back end, if you edit the action name, you lose everything you previously configured and must reconfigure the action from scratch. You must also update the action class `__init__.py` refers to.
4. Click **Finish**.
5. To download the service, select the download  icon, then download the ZIP file.
6. In Incident Responder, [upload](#) the service. You must have version i53 or later.

Delete a Custom Action in Action Editor

Delete a custom action you [created](#) in Action Editor, then re-upload the package to Incident Responder

1. Select the custom service, hover over the custom action, then select the More  menu.
2. Select **Delete Action**, then click **Delete**.
3. To download the service, select the download  icon, then download the ZIP file.
4. In Incident Responder, [upload](#) the service. You must have version i53 or later.