Contact Information

Forescout Technologies, Inc.
190 West Tasman Drive
San Jose, CA 95134 USA
https://www.Forescout.com/support/
Toll-Free (US): 1.866.377.8771
Tel (Intl): 1.408.213.3191
Support: 1.708.237.6591

About the Documentation

- Refer to the Technical Documentation page on the Forescout website for additional documentation: https://www.Forescout.com/company/technical-documentation/
- Have feedback or questions? Write to us at documentation@forescout.com

Legal Notice

© 2020 Forescout Technologies, Inc. All rights reserved. Forescout Technologies, Inc. is a Delaware corporation. A list of our trademarks and patents can be found at https://www.Forescout.com/company/legal/intellectual-property-patents-trademarks. Other brands, products, or service names may be trademarks or service marks of their respective owners.
# Table of Contents

## About Connect Plugin .......................................................................................... 5

## Connect User Interface Overview ........................................................................ 7

### Build an App with Connect ............................................................................. 13
- Define system.conf File .................................................................................. 13
  - Define Name, Version, and Author in system.conf ..................................... 15
  - Define User Interface Panels and Fields in system.conf ............................. 17
- Summary of system.conf Rules ......................................................................... 34
- Define property.conf File ............................................................................... 35
  - Define Name in property.conf .................................................................. 36
  - Define Property Groups in property.conf .................................................... 36
  - Define Properties in property.conf ............................................................ 37
  - Define Action Groups in property.conf ....................................................... 44
  - Define Actions in property.conf ................................................................ 45
  - Map Scripts in property.conf ..................................................................... 50
  - Define Policy Templates in Connect ........................................................... 51
  - Define Policy Template Group in property.conf ........................................... 52
  - Define Policies in property.conf ................................................................ 53
  - Define Icons in Connect .......................................................................... 54
- Create Policy Template XML File for Connect ................................................... 56
- Write Python Scripts for Connect ................................................................. 57
  - About Python Scripting for Connect .......................................................... 58
  - Test Script for Connect ............................................................................ 59
  - Polling Script for Connect ........................................................................ 60
  - Action Script for Connect ......................................................................... 61
  - Property Resolve Script for Connect .......................................................... 63
  - Authorization Script for Connect ............................................................... 64
  - Use App Instance Cache in Connect Scripts ................................................. 65
  - Use Certificate Validation in Connect Scripts .............................................. 65
- Create a Connect App ......................................................................................... 65

### Deploy an App with Connect ........................................................................... 67
- Download a Connect App from GitHub ............................................................ 67
- Install Connect Plugin ....................................................................................... 70
- Connect Add-On Module (Optional) .................................................................. 71

## Connect User Interface Details .......................................................................... 73
- Connect Pane Details ....................................................................................... 74
  - Columns in Connect Pane .......................................................................... 74
  - Buttons in Connect Pane ............................................................................ 76
  - Menu in Connect Pane ................................................................................ 76
- System Description Dialog Box Details ............................................................ 84
  - Columns in System Description Dialog Box ................................................ 85
  - Buttons in System Description Dialog Box .................................................. 86
  - Menu in System Description Dialog Box ...................................................... 98
- Configure Policy Templates in Connect ........................................................... 99
Appendix A: Sample Connect Files

Sample system.conf File
Sample property.conf File
Sample Policy Template .xml File for Connect
Sample Connect Script Files
Sample Test Script for Connect
Sample Polling Script for Connect
Sample Resolve Script for Connect
Sample App Instance Cache Script for Connect
Sample Add a User Action Script for Connect
Sample Delete a User Action Script for Connect
Sample Authorization Script for Connect
About Connect Plugin

The Connect Plugin provides an infrastructure for integrating third-party vendors with the Forescout platform. Use Connect to create third-party vendor integrations quickly. Use this guide to build and deploy applications.

Audience

The audience for Connect is technical people who want to create third-party vendor integrations, such as:

- Forescout users who want to build custom integrations
- Forescout experts
- Forescout partners
- Third-party developers who want to build integrations with Forescout

The audience needs the following:

- Knowledge of the Forescout platform
- Beginner to intermediate skills with Python scripting
- Knowledge of RESTful API concepts

The following knowledge may also be helpful:

- Forescout platform’s Open Integration Module (OIM)
- Third-party vendor APIs

This guide also supports a non-technical user, one who configures the applications built by the technical audience.

About this Guide

This guide has two parts. Build an App with Connect describes how to define a third-party vendor integration with Connect. It is intended for app developers and describes how to create an app.

Deploy an App with Connect describes how to deploy an app. It is intended for app users and describes downloading and installing an app, licensing, and configuring. The Connect User Interface Overview and Connect User Interface Details are also relevant topics for this user.

What an App Builder Does

An app builder does the following:

- Defines content in configuration files for the system description, which is the connection to the third-party vendor, as well as the properties, actions, and policy templates for the integration
- Writes Python scripts to accomplish tasks in the Forescout platform such as resolve properties, take actions, or poll for endpoints
- Creates an application (app) by putting the configuration files and Python scripts in a zip file
- Imports the app into the Connect Plugin
The result is a user interface in which an app user configures the connection and the policies or takes actions in Connect for that integration.

**Supported Forescout Platform Version**

The following table lists the Forescout platform version that works with each version covered by this guide.

<table>
<thead>
<tr>
<th>Version</th>
<th>Forescout Platform Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.0</td>
<td>Minimum version: 8.1.4 or 8.2.1</td>
</tr>
<tr>
<td>1.4.1</td>
<td>Minimum version: 8.1.4 or 8.2.1</td>
</tr>
</tbody>
</table>

**Customer Support**

The Connect Plugin is supported by Forescout Customer Support. See https://forescout.force.com/support/s/.

Connect Apps, including those provided by Forescout, are not supported by Forescout.

**Architecture**

The Connect Plugin lets you import apps that solve specific use cases for a third-party integration. The architecture has the following components.

- **Apps**
- **Python Server**
- **eyeExtend Connect**
- **Forescout Platform**

At the base of the architecture is the Forescout platform infrastructure, including the Forescout Console and CounterACT® Appliances.

The Connect Plugin software is deployed on a CounterACT Appliance and installed with a .fpi file, similar to other plugins or eyeExtend modules.

A Python server is included in the .fpi file. The server is needed to run the scripts in an app. The Python server runs on a CounterACT Appliance and is shared by all apps.

Apps contain the configuration files and Python scripts for a specific integration.
About Apps

Apps contain the following:

- **System configuration file (system.conf)**: a configuration file in JSON format that contains information that a user would need to configure an integration. The system.conf file determines the panels and fields that are displayed in the Connect pane in the Forescout Console. For example, the system.conf file might specify that an integration needs an Add Connection panel with fields for a URL, username, and password, an Assign CounterACT Devices panel, and a Proxy Server panel.

- **Property configuration file (property.conf)**: a configuration file in JSON format that contains properties specific to the integration you want to create. The property.conf file also defines actions, policy templates, and maps scripts. Script mapping ties the properties and actions in the property.conf file to the Python scripts.

- **Python script**: a script file to solve a specific use case. There can be multiple scripts. For example, one script might resolve properties, another might take actions, and a third might poll for endpoints.

In this guide, the system configuration file and property configuration file are referred to as system.conf and property.conf, however, system.json and property.json are also accepted file names. The suffixes do not have to match, for example, an app can have a property.conf and a system.json file.

At a minimum, an app contains three files:

- system.conf
- property.conf
- one Python script

All the files are put in a zip file to be imported into Connect.

Connect User Interface Overview

This topic provides an overview of the user interface.

After Connect is installed, Connect is displayed under Options. See Install Connect Plugin.
**Connect Pane Overview**

Initially, the *Connect* pane is blank. No apps have been imported yet and no system descriptions have been configured yet.

There are several buttons on the *Connect* pane:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td>Import an app</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit an app</td>
</tr>
<tr>
<td>Update</td>
<td>Update an app</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove an app</td>
</tr>
<tr>
<td>Start</td>
<td>Start an app</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop an app</td>
</tr>
</tbody>
</table>

The *Import* and *Edit* buttons are briefly described in this overview. All the buttons are described in *Connect User Interface Details*. 
Select **Import** to import apps into Connect. Apps are in zip or eca format. They can be in any folder. The following example shows a .zip file, which contains the files for the app.

Apps that have been signed by Forescout are in a .eca file, which contains a data.zip file and a signature file.

The data.zip file contains the files for the app.

After an app is imported, the **System Description** dialog box opens. It is initially blank. See **System Description Dialog Box Overview** for details.
After the system description for an app is configured, it is displayed in the **Connect** pane. There can be multiple apps displayed in this pane.

- Third-party vendor integrations are displayed inside the **Connect** pane, not on the left under **Options**.

If the configuration has not been saved, select **Apply** to enable the **Start** button, which starts an app and the **Stop** button, which stops an app.

You can select an existing app and then select **Edit** to open the **System Description** dialog box.

**System Description Dialog Box Overview**

If no system descriptions have been configured yet, the **System Description** dialog box is blank.
There are several buttons for a system description as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Add a system description</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit a system description</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove a system description</td>
</tr>
<tr>
<td>Test</td>
<td>(Optional) Test a system description</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refresh app instance cache data</td>
</tr>
<tr>
<td>Import</td>
<td>Import a system description</td>
</tr>
<tr>
<td>Export</td>
<td>Export a system description</td>
</tr>
</tbody>
</table>

This overview describes the **Add** button. The other buttons are described in [Connect User Interface Details](#).

Select **Add** on the **System Description** dialog box to add a system description, which defines a connection to a third-party vendor. The system.conf file determines the configuration panels and the fields on each panel that can be configured when you select **Add**.
The user configuring the system description enters the information on the panel. Select **Next** to display the next configuration panel that is defined in the system.conf file. There can be multiple panels. Then select **Finish**.

After a system description is configured, it is displayed in the **System Description** dialog box.
Build an App with Connect

This topic describes how to define a third-party vendor integration with Connect. It is intended for app developers and describes how to create an app. See the following:

- Define system.conf File
- Define property.conf File
- Create Policy Template XML File for Connect
- Write Python Scripts for Connect
- Create a Connect App

Define system.conf File

The system configuration or system.conf file contains information that a user needs to configure an integration. The file is in JSON format. You can use any text editor to edit it, such as Notepad++.

The system.conf file determines the panels and fields that are displayed in the user interface. For example, the system configuration file might specify that an integration needs a Connection panel with several fields, an Assign CounterACT Devices panel, and a Proxy Server panel.

The system configuration file must be named either system.conf or system.json. It has the following sections:

- App information, such as name, version, and author. See Define Name, Version, and Author in system.conf.
- Definitions of the panels and the fields in the user interface. See Define User Interface Panels and Fields in system.conf.
The following is a sample system.conf file (split into two parts). Four panels have been defined. The first panel has been defined with seven fields.

```json
{
    "name": "Cylance",
    "version": "1.0.0",
    "extractable": true,
    "panels": {
        "Cylance Connection": {
            "title": "Cylance Connection",
            "description": "Cylance Connection",
            "fields": [
                {
                    "display": "URL",
                    "field ID": "connect_cylance_url",
                    "type": "string",
                    "mandatory": true,
                    "add to column": true,
                    "show column": "true",
                    "identifier": "true",
                    "tooltip": "URL"
                },
                {
                    "display": "Tenant ID",
                    "field ID": "connect_cylance_tenant_id",
                    "type": "string",
                    "mandatory": true,
                    "add to column": true,
                    "show column": "false",
                    "tooltip": "Tenant ID"
                },
                {
                    "display": "Application ID",
                    "field ID": "connect_cylance_application_id",
                    "type": "string",
                    "mandatory": true,
                    "add to column": true,
                    "show column": "false",
                    "tooltip": "Application ID"
                },
                {
                    "display": "Application Secret",
                    "field ID": "connect_cylance_application_secret",
                    "type": "string",
                    "mandatory": true,
                    "tooltip": "Application Secret"
                },
                {
                    "certification validation": true,
                    "app_instance_cache": true,
                    "display": "Custom configuration refresh interval (in minutes)",
                    "min": 1,
                    "max": 2400,
                    "value": 300
                },
                {
                    "authorization": true,
                    "display": "Authorization Interval (in minutes)",
                    "min": 1,
                    "max": 100,
                    "value": 10
                }
            ]
        }
    }
}
```
Define Name, Version, and Author in system.conf

Define “name”, “version”, and “author” in the system.conf file, all of which are required fields. There is also a field for an optional, predefined Test button.

{
    "name":"appname",
    "version":"1.0.0",
    "author":"FirstName LastName",
    "testEnable":true,
}

Name, Version, and Author in User Interface

The name, version, and author defined in the system configuration file are displayed in the user interface on the Connect pane.
Test Button in User Interface

If enabled, the Test button is displayed on the System Description dialog box. The Test button can be used to test connectivity to a third-party vendor through a Python script.

Parameter Details for Name, Version, Author, and Test Button

The parameters in the system configuration file are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>(Required) The name of the app. Each app must have a unique name. The name in the system.conf and property.conf files must match. The name can contain letters (uppercase and lowercase), numbers, spaces, and special characters, but not the underscore character (_). Names are used in various places in configuration files and Python scripts. Select a specific name rather than a generic name.</td>
</tr>
<tr>
<td>&quot;version&quot;</td>
<td>(Required) The version of the app. The format of the version can be one, two, or three integers with a period as a separator, for example: 1, 1.0, 1.0.1</td>
</tr>
<tr>
<td>&quot;author&quot;</td>
<td>(Required) The author of the app. The author can contain letters (uppercase and lowercase), numbers, spaces, and special characters.</td>
</tr>
<tr>
<td>&quot;testEnable&quot;</td>
<td>(Optional) The device test button. If set to true, a Test button is displayed in the System Description dialog box.</td>
</tr>
</tbody>
</table>

Edit Name, Version, and Author

To edit the system.conf file:

1. Specify a name, version, and author by entering the text within the quotation marks, for example:
   
   "name":"Cylance ",
   "version":"1.0.0",
2. (Optional) To display the **Test** button on the **System Description** dialog box, keep the following:

   "testEnable":true,

To not display the **Test** button on the **System Description** dialog box, either replace true with false or remove “testEnable” from the system.conf file.

**Define User Interface Panels and Fields in system.conf**

The rest of the system.conf file defines the configuration panels and fields needed in the user interface to define the system description of the integration.

In the following sample configuration (split into two parts), there are four panels. Two of the panels (the second and third) are predefined. The first panel is a custom panel with seven fields. The first four fields on that panel are defined using several parameters, while the fifth, sixth, and seventh fields on that panel are predefined. The fourth panel has two predefined fields.
Panels and Fields in User Interface

When you select **Add** in the **System Description** dialog box, the first configuration panel defined in the system.conf file is displayed. For example, the following panel has several fields as well as a Validate Server Certificate checkbox.
When you select **Next**, the second configuration panel defined in the system.conf file is displayed, for example, the predefined Assign CounterACT Devices panel.

When you select **Next**, the third configuration panel defined in the system.conf file is displayed, for example, the predefined Proxy Server panel.

See “rate limiter” Field and “host discovery” Field for the fourth panel defined in this sample configuration.

When you select **Finish**, the system description is configured.
Parameter Details for Panels and Fields

The parameters for "panels" and "fields" in the system configuration file are as follows:

<table>
<thead>
<tr>
<th>Panel</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;panels&quot;</td>
<td></td>
<td>(Required) The definition of the configuration panels in the user interface. In this sample configuration, there are four panels:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ &quot;title&quot;:&quot;Cylance Connection&quot;—a custom panel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ &quot;focal appliance&quot;:true—a predefined panel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ &quot;proxy server&quot;:true—a predefined panel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ &quot;title&quot;:&quot;Cylance Options&quot;—a custom panel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value is a JSON array. Each element of the JSON array is a JSON object.</td>
</tr>
<tr>
<td>&quot;title&quot;</td>
<td></td>
<td>(Required) The title of a custom panel.</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td></td>
<td>The description of a custom panel.</td>
</tr>
</tbody>
</table>
### Panel Field Description

<table>
<thead>
<tr>
<th>Panel</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
|      | "fields" | (Required) The fields of a custom panel. In this sample configuration there are several fields that start with "display", as well as predefined fields. The value is a JSON array. Each element of the JSON array is a JSON object. Use the following parameters to define fields:  
  - "display"—(Required) The label of the field, which will be displayed on the left of the field in the user interface, such as URL or Proxy Server Port.  
  - "field ID"—(Required) The internal, unique name of the field, in ASCII characters. It must start with `connect_<appname>`. The `<appname>` must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the `<appname>` for VMware AirWatch would be `vmwareairwatch`. Field ID values are global variables.  
  - "type"—(Required) The type of the field. The valid types are: shortString, longString, ip, integer, boolean, encrypted, and option. See "type" Parameter Details.  
  - "mandatory"—(Required) When set to true, the field is mandatory in the user interface, which means that when it is configured, the field must not be empty. See "mandatory" Parameter.  
  - "add to column"—When set to true, you can select the field from the Add/Remove Columns dialog box. You cannot set this parameter to true if the field type is encrypted. See "add to column" Parameter in User Interface.  
  - "show column"—When set to true, you can display the field as a column in the System Description dialog box, if "add to column" is also set to true. See "show column" Parameter in User Interface.  
  - "identifier"—When set to true, the value of the field must be unique. For example, for a URL. See Error Message for "identifier" Parameter in User Interface.  
  - "tooltip"—The text for the tooltip in the user interface.  
  - "value"—The value for a field that is prepopulated with a default, such as an actual URL. For example, to prepopulate the Cylance URL, use: "value":"https://protectapi.cylance.com"  

The predefined fields are:  
  - "certification validation"—When set to true, the predefined Validate Server Certificate checkbox is displayed. See "certification validation" Field.  
  - "app_instance_cache"—When set to true, the predefined refresh interval field for app instance cache data is displayed. See "app_instance_cache" Field.  
  - "authorization"—When set to true, the predefined Authorization Interval field is displayed. See "authorization" Field. |
Define Panels and Fields

Define at least two panels with one field on each panel by replacing the text in quotation marks for “panels” and “fields” in the sample system.conf file. See Sample system.conf File.

In the user interface, there will be at a minimum, a Next button on the first panel and a Finish button on the second panel.

It is recommended that you put the most important fields in the first panel, such as IP address, URL, or username/password.

You can define many parameters for each field. Four parameters for each field are required: “display”, “field ID”, “type”, and “mandatory”.

To define four fields on a panel, you need four field definitions in the system.conf file.

### Panel | Field | Description
--- | --- | ---
"focal appliance" | (Required) The predefined focal appliance panel. If set to true, a focal appliance panel is displayed as a configuration panel. It cannot be set to false because it is a required panel. See Assign CounterACT Devices Panel Details.
"title" | (Required) The title of the panel, which is the predefined focal appliance panel.
"description" | The description of the focal appliance panel. HTML formatting can be used in the description, for example, to create multiple paragraphs.
"proxy server" | (Optional) The predefined proxy server panel. If set to true, a proxy server panel is displayed as a configuration panel. See Proxy Server Panel Details.
"title" | (Required) The title of the panel, which is the predefined proxy server panel.
"description" | The description of the proxy server panel. HTML formatting can be used in the description, for example, to create multiple paragraphs.
In the following sample configuration, the Cylance Connection panel is defined with four fields. The definition of each field uses different parameters. See Parameter Details for Panels and Fields.

```
{
    "name": "Cylance",
    "version": "1.4.0",
    "author": "Connect Masters",
    "testEnable": true,
    "panels": [
        {
            "title": "Cylance Connection",
            "description": "Cylance Connection",
            "fields": [
                {
                    "display": "URL",
                    "field ID": "connect_cylance_url",
                    "type": "shortString",
                    "mandatory": true,
                    "add to column": true,
                    "show column": true,
                    "identifier": true,
                    "tooltip": "URL"
                },
                {
                    "display": "Tenant ID",
                    "field ID": "connect_cylance_tenant_id",
                    "type": "shortString",
                    "mandatory": true,
                    "add to column": true,
                    "show column": false,
                    "tooltip": "Tenant ID"
                },
                {
                    "display": "Application ID",
                    "field ID": "connect_cylance_application_id",
                    "type": "shortString",
                    "mandatory": true,
                    "add to column": true,
                    "show column": false,
                    "tooltip": "Application ID"
                },
                {
                    "display": "Application Secret",
                    "field ID": "connect_cylance_application_secret",
                    "type": "encoded",
                    "mandatory": true,
                    "tooltip": "Application Secret"
                }
            ]
        }
    ]
}
```

**“mandatory” Parameter**

It is recommended that at least one field in the first panel have the “mandatory” parameter set to true. Mandatory means that when the field is configured in the user interface, it must not be empty.

It is also recommended that you put the most important fields in the first panel, such as IP address, URL, or username/password, which are typical uses of the “mandatory” parameter.

An error message is displayed for a “mandatory” field if nothing is entered in the user interface when Next is selected.

![Error message](image.png)
**Error Message for “identifier” Parameter in User Interface**

If a field is “identifier”, the value of the field must be unique when it is configured in the user interface. For example, if the field is a URL, it must be unique.

Only one “identifier” is allowed in a system.conf file.

An error message is displayed for an “identifier” field if the same value is entered in the user interface when **Next** is selected.

![Error Message](image)

“**show column**” Parameter in User Interface

The “show column” parameter results in fields being displayed as columns in the **System Description** dialog box. In the following example, there is one column.

![System Description](image)

When “show column” is set to true, the field is displayed as a column in the **System Description** dialog box, if “add to column” is also set to true.

“**add to column**” Parameter in User Interface

The “add to column” parameter results in the following menu when you right-click in the **System Description** dialog box.

![Menu](image)
If you select **Add/Remove Columns**, you can select columns to add or remove from the Available Columns and Selected Columns tables.

Select **OK** to see the selected columns displayed in the **System Description** dialog box.

If the field type is "encrypted", you cannot add it as a column. This prevents the display of passwords in a column.

At least one field in a system description must have the “add to column” parameter set to true or an error message is displayed.

**“type” Parameter Details**

The “type” parameters are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;shortString&quot;</td>
<td>A string field, consisting of one row of editable text. Use this type for a username or a URL.</td>
</tr>
<tr>
<td>&quot;longString&quot;</td>
<td>A string field, consisting of five rows of editable text. Use this type for a multi-line field, such as a description.</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;ip&quot;</td>
<td>An IP field. Use this type for an IPv4 address.</td>
</tr>
<tr>
<td>&quot;integer&quot;</td>
<td>An integer field, which can be a number from 1 to 2,147,483,647. Use this type for a list of numbers. You can either scroll to select a number or type a number in the field.</td>
</tr>
<tr>
<td>&quot;boolean&quot;</td>
<td>A Boolean field. Use this type to create a checkbox that takes only a &quot;true&quot; (checked) or &quot;false&quot; (unchecked) value.</td>
</tr>
<tr>
<td>&quot;encrypted&quot;</td>
<td>An encrypted field, in which the values are encrypted. This field type also creates a field to verify the value. Use this type for a password.</td>
</tr>
<tr>
<td>&quot;option&quot;</td>
<td>An option field containing a drop-down menu. Use this type for a selectable list.</td>
</tr>
</tbody>
</table>

In addition to specifying the type, you also must specify the name and value of the options. The following sample "option" definition produces the user interface on the right.

```
"type": "option",
"options": [
  { "display": "option1", "value": "OPTION1_FIELD_ID" },
  { "display": "option2", "value": "OPTION2_FIELD_ID" },
  { "display": "option3", "value": "OPTION3_FIELD_ID" }
]
```

### "certification validation" Field

The "certification validation" field is predefined and results in the following checkbox on a panel in the user interface.

```
Validate Server Certificate ✔
```

Only one "certification validation" field is allowed in a system.conf file.
If this checkbox is selected during configuration, the identity of the third-party server is validated before establishing a connection. If this checkbox is not selected during configuration, certificate-based authentication is disabled and self-signed certificates are accepted.

Trusted certificates must be uploaded in the Forescout Console to **Certificates > Trusted Certificates**. Upload the entire certificate chain. For more information about certificates, refer to “Configuring the Certificate Interface” in the *Forescout Administration Guide*.

Certificates in Connect can be used by all apps.

**“app_instance_cache” Field**

The following is a sample configuration of an application instance cache field that can be defined in the system.conf file. It uses a predefined parameter named “app_instance_cache”.

```json
{
   "app_instance_cache":true,
   "display":"Custom configuration refresh interval (in minutes)",
   "min":5,
   "max":2400,
   "value":240
}
```

You can store and retrieve non-endpoint data from a third-party product (the data is not associated with an endpoint).

The value is stored as a string type of system description field, which you can save in any format, such as a JSON array. It is retrievable in property resolve, action, and discovery scripts using `params.get("connect_app_instance_cache")`.

The different types of scripts (property, action, polling) can access the data at the per configuration instance of the app.

For example, you can use this functionality to get a list of all users, then use the list in an action script. See **Use App Instance Cache in Connect Scripts**.

The predefined “app_instance_cache” field also lets you configure the interval of how often you want to refresh the data. When the field is defined, the Refresh button on the **System Description** dialog box is enabled to manually trigger a refresh. See **Refresh App Instance Cache Data**.

To enable this functionality, you also need to specify the script used for retrieving the data from the third-party. The script must be mapped in the property.conf file. See **Map Scripts in property.conf**.

Only one “app_instance_cache” field is allowed in a system.conf file.

The parameters of the app_instance_cache field are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“app_instance_cache”</td>
<td>(Required) When set to true, indicates an integer field for configuring the app instance cache data refresh interval, in minutes. Also, the Refresh button on the <strong>System Description</strong> dialog box is enabled when a system description is selected.</td>
</tr>
<tr>
<td>“display”</td>
<td>(Required) The label of the field, which will be displayed on the left of the field in the user interface.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>&quot;min&quot;</td>
<td>(Optional) The minimum value of the field. If none is set, &quot;min&quot; defaults to 1.</td>
</tr>
<tr>
<td>&quot;max&quot;</td>
<td>(Optional) The maximum value of the field. If none is set, &quot;max&quot; will be the maximum positive value for a 32-bit signed binary integer.</td>
</tr>
<tr>
<td>&quot;value&quot;</td>
<td>(Optional) The prepopulated default value of the field. If none is set, &quot;value&quot; will be the &quot;min&quot;.</td>
</tr>
</tbody>
</table>

The following is an example of the refresh interval field in the user interface:

```
Custom configuration refresh interval (in minutes) 240
```

### "authorization" Field

The following is a sample configuration of an authorization field that can be defined in the system.conf file. It uses a predefined parameter named "authorization".

```json
{
  "authorization": true,
  "display": "Authorization Interval (in minutes)",
  "min": 1,
  "max": 100,
  "value": 28
}
```

You can enable an interval-based authorization mechanism, which you can then use in all action, polling, and resolve scripts.

For the Authorization Interval field, you specify the minimum and maximum number of minutes for the interval at which the authorization is refreshed, and a default. In this sample configuration, the minimum is one minute, the maximum is 100 minutes, and the default is 28 minutes, which means the authorization is refreshed every 28 minutes.

It is recommended that you set the value of the field to less than the authorization expiry. For example, if the authorization expires every 30 minutes, but you refresh every 28 minutes, you can guarantee that you will always have a valid authorization.

To enable "authorization", a script is also needed. See Authorization Script for Connect. In addition, the authorization script must be mapped in the property.conf file. See Map Scripts in property.conf.

Only one "authorization" field is allowed in a system.conf file.

The parameters of the authorization field are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;authorization&quot;</td>
<td>(Required) When set to true, indicates an integer field for the authorization refresh interval (in minutes).</td>
</tr>
<tr>
<td>&quot;display&quot;</td>
<td>(Optional) The minimum value of the field. If none is set, &quot;min&quot; defaults to 1.</td>
</tr>
<tr>
<td>&quot;display&quot;</td>
<td>(Required) The label of the field, which will be displayed on the left of the field in the user interface.</td>
</tr>
</tbody>
</table>
The resulting field, Authorization Interval (in minutes), is displayed in the user interface.

![Authorization Interval field](image)

### “rate limiter” Field

The following is a sample configuration of a rate limiter field that can be defined in the system.conf file. It uses a predefined parameter named “rate limiter”.

```json
{
  "rate limiter": true,
  "display": "Number of API queries per unit time",
  "unit": 1,
  "min": 1,
  "max": 1000,
  "add to column": "true",
  "show column": "false",
  "value": 100
}
```

You can rate limit the requests sent to the third-party server. The rate limiter specifies the number of times a script is invoked during the specified time. It is triggered when the app starts.

For the Number of API queries per unit time field, you specify the unit to use, such as seconds, minutes, or hours, the minimum and maximum values, and a default. In this sample configuration, the unit is one second, the minimum is one second, the maximum is 1000 seconds, and the default is 100 seconds.

The script invocations are held as tasks in a queue. When the rate limiter constraint is reached, there is a wait before the script is invoked again.

The tasks, up to the number specified in the value field, will be executed from the pending queue per the specified time.

Only one “rate limiter” field is allowed in a system.conf file.

The parameters of the rate limiter field are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“rate limiter”</td>
<td>(Required) When set to true, indicates an integer field for the rate limiter.</td>
</tr>
<tr>
<td>“display”</td>
<td>(Required) The label of the field, which will be displayed on the left of the field in the user interface.</td>
</tr>
<tr>
<td>“unit”</td>
<td>(Required) The unit for the rate limiter, which is an integer. For example, a unit of 1 is 1 second, a unit of 60 is 1 minute, and a unit of 3600 is 1 hour.</td>
</tr>
<tr>
<td>“min”</td>
<td>(Optional) The minimum value of the field. If none is set, “min” defaults to 1.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
“max” | (Optional) The maximum value of the field. If none is set, “max” will be the maximum positive value for a 32-bit signed binary integer.
“add to column” | When set to true, you can select the field from the Add/Remove Columns dialog box.
“show column” | When set to false, the field is not displayed as a column in the System Description dialog box. When set to true, the field is displayed as a column in the System Description dialog box if “add to column” is set to true.
“value” | (Optional) The prepopulated default value of the field. If none is set, “value” will be the “min”.

The resulting field, Number of API queries per unit time, is displayed in the user interface.

```
"host discovery": true,
"display": "Discovery Frequency",
"max": 300000,
"add to column": true,
"show column": false,
"value": 3600
```

Only one “host discovery” field is allowed in a system.conf file.
The parameters of the host discovery field are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“host discovery”</td>
<td>(Required) When set to true, indicates a checkbox to enable and disable a host discovery field as well as an integer field for the discovery frequency.</td>
</tr>
<tr>
<td>“display”</td>
<td>(Required) The label of the field, which will be displayed on the left of the field in the user interface.</td>
</tr>
<tr>
<td>“max”</td>
<td>(Optional) The maximum value of the field. If none is set, “max” will be the maximum positive value for a 32-bit signed binary integer.</td>
</tr>
<tr>
<td>“add to column”</td>
<td>When set to true, you can select the field from the Add/Remove Columns dialog box.</td>
</tr>
<tr>
<td>“show column”</td>
<td>When set to false, the field is not displayed as a column in the System Description dialog box. When set to true, the field is displayed as a column in the System Description dialog box if “add to column” is set to true.</td>
</tr>
<tr>
<td>“value”</td>
<td>(Optional) The prepopulated default value of the field. If none is set, “value” defaults to 1. The unit is minutes.</td>
</tr>
</tbody>
</table>

The resulting Enable Host Discovery checkbox and Discovery Frequency field are displayed in the user interface.

---

**Assign CounterACT Devices Panel Details**

The “focal appliance” panel is required in the system.conf file. Host discovery, property resolve, and actions are communicated via the focal appliance to the endpoint.

You cannot set "focal appliance" to false. If it is missing from the configuration, an error message is displayed.

In general, it is not recommended to use the Enterprise Manager as the connecting CounterACT device. But if you must, make sure that it is not used to discover MAC-only hosts.
The “focal appliance” panel is predefined and results in the following panel, with **Assign all devices by default** selected, so you can add one device.

More devices can be added after the first device. Subsequently, the Assign CounterACT Devices panel has more fields. See [Add a System Description](#) for details about the predefined fields on the panel.

Note the following:

- An error message is displayed if you try to add a device that is already used.
- Only one focal appliance panel is allowed in a system.conf file.
- The focal appliance panel cannot be the first panel defined in the system.conf file.
- The focal appliance must be the managing appliance for overlapping IPs.
Proxy Server Panel Details
The “proxy server” panel is predefined and results in the following panel.

Both authentication and non-authentication modes are supported. For example, a proxy server username and password are not required.

Only one proxy server panel is allowed in a system.conf file.

See Add a System Description for details about the predefined fields on the panel.

The Proxy Server panel has predefined field IDs, which you might need to use in a Python script. The field IDs for the Proxy Server panel are:

- Use Proxy Server: connect_proxy_enable
- Proxy Server: connect_proxy_ip
- Proxy Server Port: connect_proxy_port
- Proxy Server Username: connect_proxy_username
- Proxy Server Password: connect_proxy_password

There is no field ID for the Verify Password field because the "encrypted" field type includes the verify field.

Summary of system.conf Rules
The following is a summary of the system.conf rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>For More Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the same &quot;name&quot; in the system.conf and property.conf files</td>
<td>Parameter Details for Name, Version, Author, and Test Button</td>
</tr>
<tr>
<td>Define at least two panels with one field on each panel</td>
<td>Define Panels and Fields</td>
</tr>
</tbody>
</table>
Rule | For More Information
---|---
Define at least one field on the first panel with a “mandatory” parameter set to “true” | “mandatory” Parameter
Only one “identifier” parameter is allowed | Error Message for “identifier” Parameter in User Interface
Only one ”certification validation” field is allowed | “certification validation” Field
Only one “app_instance_cache” field is allowed | “app_instance_cache” Field
Only one “authorization” field is allowed | “authorization” Field
Only one ”rate limiter” field is allowed | “rate limiter” Field
Only one “host discovery” field is allowed | “host discovery” Field
Define at least one field with “add to column” parameter set to true | “add to column” Parameter in User Interface
If the “type” is encrypted, it cannot be added as a column | “add to column” Parameter in User Interface
Focal appliance panel is required | Assign CounterACT Devices Panel Details.
Only one focal appliance panel is allowed
Focal appliance panel cannot be the first panel | Proxy Server Panel Details

Define property.conf File

The property configuration or property.conf file contains properties specific to the integration you want to create. The property.conf file also defines actions and maps scripts. Script mapping ties the properties and actions in the property.conf file to the Python scripts. The property.conf file is in JSON format. You can use any text editor to edit it, such as Notepad++.

You can also define policy templates and icons in the property.conf file.

The property configuration file must be named either property.conf or property.json. It has the following sections:

- Define Name in property.conf
- Define Property Groups in property.conf
- Define Properties in property.conf
- Define Action Groups in property.conf
- Define Actions in property.conf
- Map Scripts in property.conf
- Define Policy Templates in Connect
- Define Policy Template Group in property.conf
- Define Policies in property.conf
- Define Icons in Connect
See Sample property.conf File.

Define Name in property.conf

Define the “name” field in the property.conf file, which is a required field.

```json
{
    "name": "Cylance",
}
```

Parameter Details for Name

The parameter for “name” is as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“name”</td>
<td>(Required) The name of the app. Each app must have a unique name. The name in the system.conf and property.conf files must match. The name can contain letters (uppercase and lowercase), numbers, spaces, and special characters, but not the underscore character (_). Names are used in various places in configuration files and Python scripts. Select a specific name rather than a generic name.</td>
</tr>
</tbody>
</table>

Define Property Groups in property.conf

Use “groups” to define property group names. The value of “groups” is a JSON array. Each element of the JSON array is a JSON object.

In the following example, one group is defined, but there can be multiple.

```json
"groups": [
    {
        "name": "connect_cylance_cylance",
        "label": "Cylance"
    }
],
```

“groups” in User Interface

The “groups” parameter results in a label for the property group displayed in the user interface in the Condition dialog box.
Parameter Details for Property Groups

The parameters for “groups” are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“name”</td>
<td>(Required) The internal, unique name of the group, in ASCII characters. It must start with connect_&lt;appname&gt;_. The &lt;appname&gt; must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the &lt;appname&gt; for VMware AirWatch would be vmwareairwatch.</td>
</tr>
<tr>
<td>“label”</td>
<td>(Required) The label of the group displayed in the user interface.</td>
</tr>
</tbody>
</table>

Define Properties in property.conf

Use “properties” to define the properties that can be used as conditions in policies. The value of “properties” is a JSON array. Each element of the JSON array is a JSON object.

In the following example, one property is defined. There can be multiple properties, each with different parameters.

```
"properties": [
{
  "tag": "connect_cylance_state",
  "label": "Cylance State",
  "description": "Cylance State",
  "type": "string",
  "options": [
    { "name": "Online",
      "label": "Online" },
    { "name": "Offline",
      "label": "Offline" }
  ],
  "group": "connect_cylance_cylance",
  "resolvable": true,
  "require_host_access": false,
  "inventory": [
    { "enable": true,
      "description": "Inventory of Cylance State" },
    { "tag": "asset_portal", "value": "true" }
  ],
  "track_change": { "enabled": true,
    "label": "Cylance State Changed",
    "description": "Track Change property for cylance state" },
  "dependencies": [
    { "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
}
```
“properties” in User Interface

The “properties” parameter results in properties displayed in the user interface in the **Condition** dialog box.

Parameter Details for Properties

The parameters for “properties” are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;tag&quot;</td>
<td>(Required) The internal, unique name of the property, in ASCII characters. It must start with <code>connect_&lt;appname&gt;</code>. The <code>&lt;appname&gt;</code> must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the <code>&lt;appname&gt;</code> for VMware AirWatch would be <code>vmwareairwatch</code>. If the properties are used in a script to resolve properties, the property tags must be listed in the property.conf file under “scripts”. See Parameter Details for Scripts.</td>
</tr>
<tr>
<td>&quot;label&quot;</td>
<td>(Required) The label of the property displayed in the user interface. See “label” in User Interface.</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td>(Required) The description of the property displayed in the user interface. See “description” in User Interface.</td>
</tr>
<tr>
<td>&quot;type&quot;</td>
<td>(Required) The type of the property. The valid types are string, boolean, integer, date, and composite. See Property &quot;type&quot; Details.</td>
</tr>
<tr>
<td>&quot;group&quot;</td>
<td>(Required) The group to which the property belongs. This must match the name defined in “groups” or the name of an existing property group in the Forescout platform. See Define Property Groups in property.conf.</td>
</tr>
<tr>
<td>&quot;options&quot;</td>
<td>(Optional) The options of a string property type. See “options” in Property &quot;type&quot; Details.</td>
</tr>
<tr>
<td>&quot;resolvable&quot;</td>
<td>(Optional) When set to true, the Forescout platform requests to resolve the property through policy recheck. When set to false, there is no request for rechecking the property; it is resolved through periodic polling/host discovery. The default is true.</td>
</tr>
<tr>
<td>&quot;require_host_access&quot;</td>
<td>(Optional) When set to true, resolving the host requires open TCP or UDP ports on the host. When set to false, resolving the host does not require open TCP or UDP ports on the host. The default is false.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| "inventory"     | (Optional) When set to true, add this field as a column in the Inventory view. The default is false. The value of "inventory" is a JSON object with the following fields:  
  • "enable"—(Required) When set to true, the Inventory view is required to create the property  
  • "description"—(Required) The description of the Inventory                                                                 |
| "asset_portal"  | (Optional) When set to true, the property is displayed in the asset profile. When set to false, the property is not displayed in the asset profile. The default is true. |
| "track_change"  | (Optional) Indicates if another property needs to be created in the Track Change group, which can be used as a policy condition that identifies changes in the property value.  
  The value of "track_change" is a JSON object with the following fields:  
  • "enable"—(Required) When set to true, add track change to the property  
  • "label"—(Required) The label of the track change property in the user interface  
  • "description"—(Required) The description of the track change property                                                                 |
| "dependencies" | (Optional) Indicates if resolving this property requires values of other properties. Use "dependencies" to add a dependent field to a property. When resolving this property, the value of the dependent property is sent to the Python script.  
  The value of "dependencies" is a JSON array. Each element of the JSON array is a JSON object with the following fields:  
  • "name"—(Required) The tag name of the dependent property in the Forescout platform. For example, a MAC address could be a dependency that is used in a Python script to resolve a property.  
  • "redo_new"—(Optional) When set to true, the property needs to be resolved again when the dependent property has a value for the first time. The default is false.  
  • "redo_change"—(Optional) When set to true, the property needs to be resolved again when the value of the dependent property changes. The default is false. |
| "list"          | (Optional) When set to true, the property is a list. When set to false, the property is a single string, integer, boolean, or date. The default is false. |
| "overwrite"     | (Optional) When set to true, the property is a simple list and the old property values are replaced by new resolved values every time, if "list is also set to true.  
  When set to false, the property is a list and the new resolved values are appended to the old values every time. The default is false. |
"label" in User Interface
The "label" of a property is displayed in the **Condition** dialog box in the user interface.

![Condition dialog box](image)

"description" in User Interface
The "description" of a property is displayed in the **Condition** dialog box in the user interface.

![Condition dialog box](image)

Property “type” Details
The property “type” parameters are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Sub-Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“string”</td>
<td></td>
<td>A string type of property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Spped</em></td>
</tr>
<tr>
<td>“options”</td>
<td></td>
<td>A sub-type of the string property. Use “options” to define multiple options so that a user can select a value instead of typing it. The value of “options” is a JSON array. Each element of the JSON array is a JSON object with the following fields:</td>
</tr>
<tr>
<td>Type</td>
<td>Sub-Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>“name”</td>
<td>(Required) The value of the option used to resolve the property.</td>
</tr>
<tr>
<td></td>
<td>“label”</td>
<td>(Required) The label in the user interface.</td>
</tr>
<tr>
<td>“boolean”</td>
<td></td>
<td>A Boolean type of property.</td>
</tr>
<tr>
<td>“integer”</td>
<td></td>
<td>An integer type of property.</td>
</tr>
<tr>
<td>“date”</td>
<td></td>
<td>A date type of property.</td>
</tr>
<tr>
<td>“composite”</td>
<td>“subfields”</td>
<td>A type of property that has multiple fields. The value of “subfields” is a JSON array. Each element of the JSON array is a JSON object with the following fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “tag”—(Required) The tag of the field, which is a text string, in ASCII characters. It must be unique only within the property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “label”—(Required) The label in the user interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “description”—(Required) The description in the user interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “type”—(Required) The type of the subfield. The valid types are string, integer, boolean, and date.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “inventory”—(Optional) When set to true, add this field as a column in the Inventory view. If this subfield needs an Inventory column, “inventory” must be enabled in this property first. The default is false.</td>
</tr>
</tbody>
</table>

See “composite” Details.

“options” Details

The following sample “options” definition produces the user interface on the right in the **Condition** dialog box.

```json
{
    "type": "string",
    "options": [
        {
            "name": "Online",
            "label": "Online"
        },
        {
            "name": "Offline",
            "label": "Offline"
        }
    ]
}
```
“composite” Details

The following sample “composite” definition produces the user interface on the right in the **Condition** dialog box.

```
"type": "composite",
"group": "connect_cylance_cylance",
"inventory": {
  "enable": true,
  "description": "Inventory of Cylance Policy"
},
"subfields": [
  {
    "tag": "id",
    "label": "ID",
    "description": "Policy ID",
    "type": "string",
    "inventory": true
  },
  {
    "tag": "name",
    "label": "Name",
    "description": "Policy Name",
    "type": "string",
    "inventory": true
  }
]
```

To add the property to the Inventory view, enable inventory on the property as well as on the subfields of a composite property:

```
"type": "composite",
"group": "connect_cylance_cylance",
"inventory": {
  "enable": true,
  "description": "Inventory of Cylance Policy"
},
"subfields": [
  {
    "tag": "id",
    "label": "ID",
    "description": "Policy ID",
    "type": "string",
    "inventory": true
  },
  {
    "tag": "name",
    "label": "Name",
    "description": "Policy Name",
    "type": "string",
    "inventory": true
  }
]
```

Composite subfields cannot be added as dependencies. However, you can access the subfields if you add the parent composite property as a dependency.
Properties in Policy Templates

Properties in policy templates are selected from the **Condition** dialog box.

Properties can be used in a rule.

---

See [Configure Policy Templates in Connect](#) for the policy template procedure.
Define Action Groups in property.conf

Use "action_groups" to define action group names. The value of "action_groups" is a JSON array. Each element of the JSON array is a JSON object.

In the following example, one action group is defined, but there can be multiple.

```
"action_groups": [
  {
    "name": "connect_cylance_cylance",
    "label": "Cylance"
  }
]
```

"action_groups" in User Interface

The "action_groups" parameter results in a label for the action group displayed in the user interface menu when you right-click an endpoint in the All Hosts pane.

The "action_groups" parameter is also displayed in the user interface menu for Actions in the Action dialog box.

Parameter Details for Action Groups

The parameters for "action_groups" are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>(Required) The internal, unique name of the action group, in ASCII characters. It must start with connect_&lt;appname&gt;_. The &lt;appname&gt; must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the &lt;appname&gt; for VMware AirWatch would be vmwareairwatch.</td>
</tr>
<tr>
<td>&quot;label&quot;</td>
<td>(Required) The label of the action group displayed in the user interface.</td>
</tr>
</tbody>
</table>
Define Actions in property.conf

You can define actions that make API calls to the third-party vendor. Actions can be scheduled in policies.

One-time actions are supported, such as locking a device. With one-time actions, once the action is done, it is done. There is no need to cancel the action.

Continuous actions are also supported, such as sending data to a third-party server from an endpoint. Continuous actions maintain a state. To stop a continuous action, you need to cancel the action. A continuous action can be cancelled either manually or if the policy no longer applies.

Use “actions” to define actions. The value of “actions” is a JSON array. Each element of the JSON array is a JSON object.

In the following example, one action is defined, but there can be multiple.

```
"actions": [ 
  
  
  
  ],

"dependencies": [ 
  
  ],

"undo": { 
  "label": "Cancel Cylance Add User",
  "description": "Remove Added User"
}
```
"actions" in User Interface

The "actions" parameter results in a label for the action displayed in the user interface menu item when you right-click an endpoint in the All Hosts pane.

The "actions" parameter is also displayed in the user interface menu for Actions in the Action dialog box.

Parameter Details for Actions

The parameters for "actions" are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>(Required) The internal, unique name of the action, in ASCII characters. It must start with connect_&lt;appname&gt;_. The &lt;appname&gt; must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the &lt;appname&gt; for VMware AirWatch would be vmwareairwatch. If the actions are used in a script to execute actions, the action names must be listed in the property.conf file under &quot;scripts&quot;. See Parameter Details for Scripts.</td>
</tr>
<tr>
<td>&quot;label&quot;</td>
<td>(Required) The label of the action displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td>(Required) The description of the action.</td>
</tr>
<tr>
<td>&quot;group&quot;</td>
<td>(Required) The group to which the action belongs. This must match a name defined for &quot;action_groups&quot; or the name of an existing action group in the Forescout platform.</td>
</tr>
<tr>
<td>&quot;ip_required&quot;</td>
<td>(Optional) When set to true, the action cannot be taken when the IP address is missing on the host. The default is false.</td>
</tr>
<tr>
<td>&quot;threshold_percentage&quot;</td>
<td>(Optional) The threshold control on the action as a percentage. If the number of action requests reaches the threshold, the remaining actions are held while waiting for approval.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
---|---
“params” | (Optional) The parameters that a user must enter when taking the action. See "params" Parameter Details and "params" in User Interface.
“dependencies” | (Optional) Indicates if this action requires dependent fields. Use "dependencies" to add a dependent field to the action. The value of the dependent properties is sent to the Python script. The value of “dependencies” is a JSON array. Each element of the JSON array is a JSON object with the following fields:
  - “name”—(Required) The tag name of the dependent property, in the Forescout platform.
  - “redo_new”—(Optional) When set to true, the property needs to be resolved again when the dependent property has a value for the first time. The default is false.
  - “redo_change”—(Optional) When set to true, the property needs to be resolved again when the value of the dependent property changes. The default is false.
“undo” | (Optional) Indicates if the action can be canceled (for a continuous action). Use “undo” to define a cancel action. The value of “undo” is a JSON object with the following fields:
  - “label”—(Required) The label of the cancel action in the user interface
  - “description”—(Required) The description of the cancel action See “undo” in User Interface.

**"params" Parameter Details**
The “params” parameters are as follows:

| Parameter | Description |
---|---|
“name” | (Required) The unique name of the parameter, in ASCII characters. |
“label” | (Required) The label of the parameter displayed in the user interface. |
“description” | (Required) The description of the parameter. |
“type” | (Required) The type of the parameter. The valid types are string, integer, and boolean. |
“default” | (Optional) The default value of the action parameter. It is displayed when you add a new action. The only valid type is string. |
“multiline” | (Optional) When set to true, the action parameter is a string type that needs multiple lines of text. The default is false. |
“min” | (Optional) The minimum value that users can enter in the user interface for the parameter, when the type is integer. |
“max” | (Optional) The maximum value that users can enter in the user interface for the parameter, when the type is integer. |
**“params” in User Interface**

When you select an action with “params”, the user is prompted for information in the user interface.

![Specify Add User parameters](image)

**“undo” in User Interface**

The “undo” parameter results in a label for the cancel action displayed in the user interface menu item when you right-click an endpoint in the All Hosts pane.
**Actions in Policy Templates**

Actions in policy templates are selected from the **Action** dialog box.

```
Actions can be used in a rule.
```

See [Configure Policy Templates in Connect](#) for the policy template procedure.
Map Scripts in property.conf

Use “scripts” to provide the name of the script to call as well as its usage. Script mapping ties the properties and actions in the property.conf file to the Python scripts.

Each app must have at least one Python script in it. See Write Python Scripts for Connect.

The value of “scripts” is a JSON array. Each element of the JSON array is a JSON object.

Parameter Details for Scripts

The parameters for “scripts” are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“name”</td>
<td>(Required) The name of the script. You can name a script using the third-party vendor and what the script does, for example, cylance_poll.py. Since the scripts must be added to the top level of the .zip or data.zip file, there is no need to provide a path to them.</td>
</tr>
<tr>
<td>“properties”</td>
<td>When properties are listed, the script resolves properties. This parameter is required if the script resolves properties. The value is a JSON array of property tags. See “tag” in Parameter Details for Properties.</td>
</tr>
<tr>
<td>“actions”</td>
<td>When actions are listed, the script executes actions. This parameter is required if the script executes actions. The value is a JSON array of action names. See “name” in Parameter Details for Actions.</td>
</tr>
<tr>
<td>“is_cancel”</td>
<td>When set to true, the script cancels the action. This parameter is required if the script cancels an action. The value is a JSON array of action names. See “name” and “undo” in Parameter Details for Actions and see also Action Script for Connect.</td>
</tr>
<tr>
<td>“test”</td>
<td>When set to true, the script runs the test when the Test button in the user interface is selected. See “testEnable” in Parameter Details for Name, Version, Author, and Test Button and Test Script for Connect.</td>
</tr>
<tr>
<td>“discovery”</td>
<td>When set to true, the script discovers hosts from a third-party.</td>
</tr>
<tr>
<td>“authorization”</td>
<td>When set to true, the script gets authorizations.</td>
</tr>
<tr>
<td>“app_instance_cache”</td>
<td>When set to true, the script gets app instance cache data.</td>
</tr>
<tr>
<td>“library_file”</td>
<td>When set to true, you can put your own Python files to serve as library files within an app. See Library Files.</td>
</tr>
</tbody>
</table>
“scripts” Details

The following sample “scripts” in the property.conf file shows the mapping of seven different scripts.

```
"scripts": {
    "name": "cylance_resolve.py",
    "properties": {
        "connect_cylance_fast_loggin_user",
        "connect_cylance_last_loggin_in_user",
        "connect_cylance_mac_addresses",
        "connect_cylance_is_safe",
        "connect_cylance_id"
    },
    "name": "cylance_add_users.py",
    "actions": {
        "connect_cylance_add_user"
    },
    "name": "cylance_delete_user.py",
    "is_cancel": true,
    "actions": {
        "connect_cylance_delete_user"
    },
    "name": "cylance_test.py",
    "test": true,
    "name": "cylance_poll.py",
    "discovery": true,
    "name": "cylance_authorization.py",
    "authorization": true,
    "name": "cylance_users.py",
    "app_instance_cache": true
}
```

Define Policy Templates in Connect

If you need policy templates for your app, read Create Policy Template XML File for Connect to gain an understanding of the steps before returning to this section.

If you do not have a need for policy templates, you can skip this section.

Use “policy_template” to define policy templates.

The value of “policy_template” is a JSON object with the following fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“policy_template_group”</td>
<td>(Required) The policy template group in the app. See Define Policy Template Group in property.conf.</td>
</tr>
<tr>
<td>“policies”</td>
<td>(Required) The policy templates in the app. See Define Policies in property.conf.</td>
</tr>
</tbody>
</table>
Define Policy Template Group in property.conf

Use “policy_template_group” to define the policy template group. The value of “policy_template_group” is a JSON object.

Only one policy template group can be defined.

```
"policy_template": {
  "policy_template_group": {
    "name": "connect_cylance",
    "label": "Cylance",
    "display": "Cylance",
    "description": "Cylance templates",
    "full_description": "<html>Use Cylance policy templates to manage devices in a Cylance environment:
<li>Detect devices that are compliant.</li></ul></html>",
    "title_image": "connect_cylance.png",
  },
},
```

"policy_template_group“ in User Interface

The “policy_template_group” parameter results in a label for the policy template group displayed in the user interface.

Parameter Details for Policy Template Group

The parameters for policy template group are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>(Required) The internal, unique name of the policy template group, in ASCII characters. It must be named connect_&lt;appname&gt;. The &lt;appname&gt; must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the &lt;appname&gt; for VMware AirWatch would be vmwareairwatch.</td>
</tr>
<tr>
<td>&quot;label&quot;</td>
<td>(Required) The label of the policy template group displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;display&quot;</td>
<td>(Required) The heading of the policy template group displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td>(Optional) The description of the policy template group displayed in the user interface.</td>
</tr>
</tbody>
</table>
Define Policies in property.conf

You can customize a third-party vendor integration by defining policy templates. Use “policies” to define the policy templates. The value of “policies” is a JSON object.

In the following example, one policy template is defined, but there can be multiple.

```
"policies": [
  {
    "name": "connect_cylance_compliant",
    "label": "Cylance Compliant",
    "display": "Cylance Compliant",
    "help": "Cylance Compliant Policy",
    "description": "Creates Cylance compliant policies",
    "title_name": "CylanceCompliance.xml",
    "full_description": "Use this policy template to detect corporate hosts that are compliant.\n\n<title_image>\n\nCylance.png"
  }
]
```

"policies" in User Interface

The "policies" parameter results in policy templates in the user interface.

Parameter Details for Policy Templates

The parameters for policy templates are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;name&quot;</td>
<td>(Required) The internal, unique name of the policy template, in ASCII characters. It must start with <code>connect_&lt;appname&gt;</code>_. The &lt;appname&gt; must consist of all lowercase letters of the name defined in the app without any spaces or underscores. For example, the &lt;appname&gt; for VMware AirWatch would be vmwareairwatch.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&quot;label&quot;</td>
<td>(Required) The label of the policy template displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;display&quot;</td>
<td>(Required) The heading of the policy template displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;description&quot;</td>
<td>(Optional) The description of the policy template displayed in the user interface.</td>
</tr>
<tr>
<td>&quot;help&quot;</td>
<td>(Optional) The help information of the policy template, which is displayed when you hover the mouse over the Help button in the user interface.</td>
</tr>
<tr>
<td>&quot;file_name&quot;</td>
<td>(Required) The name of the .xml file containing the policy template. The .xml file must be saved in the app in the following path: policies/nptemplates. See App Folder Paths.</td>
</tr>
<tr>
<td>&quot;full_description&quot;</td>
<td>(Optional) The full description of the policy template displayed in the user interface. HTML formatting can be used in the full description, for example, to create a bulleted list.</td>
</tr>
<tr>
<td>&quot;title_image&quot;</td>
<td>(Optional) The icon image for the policy template. See Define Icons in Connect.</td>
</tr>
</tbody>
</table>

**Define Icons in Connect**

You can customize an app by adding icons to help identify a third-party integration visually. You can add icons for actions, action groups, property groups, and policy templates. The only valid format for icons is .png.

You must put the icons in specific folders in the zip file of the app. See App Folder Paths.

![Folder Contents](image.png)

The image names for policy templates must be specified in the property.conf file, for example:

```
"title_image": cylance.png
```
For policy template icons, the folder name and the image name (the .png file) must be the same as the name defined in the property.conf file for “policy_template_group”. To display the policy template group icon in the user interface, use the following naming convention for the icon in the zip file:

\(<\text{appname}\>/\text{images/np_ng/templatedirs/connect_<appname>/connect_<appname>.png\>

For example, in the folder, cylance/images/np_ng/templatedirs/connect_cylance, the image must be named, connect_cylance.png.

For other policy template icons, put them in the folder:

\(<\text{appname}\>/\text{images/np_ng/templatedirs/connect_<appname>\>

For action icons, if the action name is connect_cylance_add_user, then all icons regarding this action should be put in the following folder:

\(<\text{appname}\>/\text{images/np_ng/actions/connect_cylance_add_user\>

For action icons, the icon names have a specific format. For example, if the action icon is named connect_cylance_add_user.png, the icon for the corresponding failed action must have the same name but be prefixed with \texttt{failed\_}. Similarly, use the prefixes \texttt{gray\_} and \texttt{waiting\_} for those action states.

For action group icons, each group defined in the property.conf file must have a folder in the action_groups folder. The folder name and image name should be the same as the name defined in the property.conf file for “action_groups”.

For property group icons, each group defined in the property.conf file must have a folder under the field_groups folder. The folder name and image name must be the same as the name defined in the property.conf file for “groups”.

An icon has specific dimensions. They must be 16 x 16 pixels.

Create Policy Template XML File for Connect

A policy template file, in .xml format, needs to be created before you can complete the policy template definition in the property.conf file.

If you have three different policy templates, you will need three policy .xml files. See Sample Policy Template .xml File for Connect for a sample of an .xml file. The .xml files are added to the zip folder of the app.
To create policy templates:
1. Define the properties and actions in the property.conf file that will be needed by your policies.
2. Import the app containing those properties and actions into Connect.
3. In the Forescout Console, go to Policy.
4. Create a custom policy template by selecting Custom and then:
   a. Select Add.
   b. Name the policy template based on your app name and select Next.
   c. Select All IPs for the IP Address Range. (Even if you add a segment, it is not saved.) Select Next.
   d. In the Main Rule pane, select properties and actions, and then select Next.
   e. (Optional) In the Sub-Rules pane, select properties and actions, and then select Next.
   f. Select Finish.
5. When the custom policy template is created, it is displayed in the Policy Manager. Select it and then select Export. Select the .xml file format.
6. Once you have the .xml file, put it in the app under the policies/nptemplates folder. See App Folder Paths.
7. (Optional) Add an icon for the policy template group and policy template. See Define Icons in Connect.
8. Return to the property.conf file to define the policy template. See Define Policy Templates in Connect.
9. Import the app into Connect and select Apply.

Write Python Scripts for Connect

You can write scripts to accomplish tasks with Connect such as resolve properties, take actions, or poll for endpoints. Scripts communicate with the third-party vendor.

Name a script by prepending the name of the third-party vendor to what the script does. For example, if the app name is cylance and there is a polling script, poll.py, the script must be named cylance_poll.py.

The name of the script must be included in the property.conf file. See Map Scripts in property.conf.

Examples of the scripts that can be written are:

- Testing. See Test Script for Connect.
- Polling/host discovery. See Polling Script for Connect.
- Actions. See Action Script for Connect.
- Property resolve. See Property Resolve Script for Connect.
Examples of using predefined fields in scripts are:

- App instance cache. See Use App Instance Cache in Connect Scripts.
- Validate Certification. See Use Certificate Validation in Connect Scripts.

An app must have at least one script in it. For sample scripts, see Sample Connect Script Files.

**About Python Scripting for Connect**

See the following sections for information about Python scripting:

- Libraries
- Python Debug Levels
- Python Log Location
- Script Not Found

**Libraries**

The Python version that is installed with Connect is 3.6.3. It has built-in libraries that you can import. Due to security implications, do not import other libraries.

The following link contains the standard built-in libraries that can be imported into scripts:

https://docs.python.org/3.6/library/

The list of third-party libraries that are installed and ready to use are:

- pyjwt (imported as jwt)
- cryptography
- requests
- prometheus_client
- httpsig

The list of built-in libraries that are not permitted are:

- os
- subprocess
- sys
- shutil
- pathlib
- glob

The list of built-in functions that are not permitted are:

- exec
- eval
- open
You cannot import an app into the Forescout Console if it contains a script that uses an unsupported library.

Note the following:
- Calling one Python script from another Python script is not supported.
- Every script has a response object.
- All scripts are run with non-root permissions.

**Library Files**

You can put your own Python files to serve as library files within an app. Other scripts can use the methods and/or objects in these files.

In the property.conf file, under "scripts", you must name the file and set "library_file" to be true. See [Map Scripts in property.conf](#).

Scripts that use library files defined in the property.conf file must not include the import statement that refers to these library files because they have already been dynamically loaded when the app was imported.

**Python Debug Levels**

The Python debug levels match the five debug levels set in the Forescout platform using the `fstool` command. The five debug levels are: critical, error, warning, info, and debug, which have values 1 to 5 respectively.

For example:

```
fstool connect_module debug 5
```

A level higher than five will default to five. Any change to the log levels takes a few minutes to propagate.

Log level settings apply to all Connect apps.

**Python Log Location**

The Python server logs are in the following directory:

_/usr/local/forescout/plugin/connect_module/python_logs_

**Script Not Found**

The Python scripts included in the "scripts" section of the property.conf file must be present in the app. If a script is not found, there will be an exception in the Python server log.

**Test Script for Connect**

For the **Test** button in the **System Description** dialog box, you can write what you want to test in a script. The script can leverage input parameters from the user interface, such as a URL.

**To create a test script:**

1. Write the test script and put it in the top level of the zip file of the app.
2. Specify the test script name in the property.conf file. See Map Scripts in property.conf.

3. Specify "testEnable":true in the system.conf file.

The app must be saved before selecting Test in the user interface. Select OK in the System Description pane and then select Apply in the Connect pane to save the system description configuration.

For example, the Cylance testing script displays success when it gets the JSON authorization token. Test scripts for other apps may have different criteria for success or failure.

See Sample Test Script for Connect.

Response Objects

The response objects for test scripts are as follows.

**Mandatory Fields**

"succeeded": boolean to denote if the test succeeded

"result_msg" (only mandatory if "succeeded" is false) : string message to display test results

**Optional Fields**

"result_msg" (optional for if succeeded is True)

**Examples**

```json
response = {"succeeded" : True}
response = {"succeeded" : False, "result_msg" : "Test failed due to connection error."}
```

Polling Script for Connect

You can write a script in which an app polls a third-party integration to get regular updates, such as for host discovery. The poll can take place at a scheduled frequency.

The polling script needs to include a JSON array of endpoints.

The properties object for polling scripts can have scalar, list, composite, or list of composite properties.

See Sample Polling Script for Connect.

Response Objects

The response objects for polling/discovery scripts are as follows.

**Mandatory Fields**

"endpoints": a list that will contain information about new endpoints

"error" : Error message if polling script fails for a known reason

**Mandatory Sub-Fields for Endpoints**

"mac" and/or "ip": Must contain MAC and/or IP address as a string
Optional Sub-Fields for Endpoints

"properties": a map/dictionary that contains host properties; the key will be the property name and the value will be the property value

Examples

response =
{"endpoints":
[
  {
    "mac": "001122334455",
    "properties":
    {
      "property1": "property_value", "property2": "property_value2"
    }
  },
  {
    "ip": "10.1.1.1",
    "properties":
    {
      "property1": "property_value"
    }
  },
  {
    "mac": "112233445566",
    "ip": "10.2.2.2",
    "properties":
    {
      "property1": "property_value"
    }
  }
]
}

Action Script for Connect

You can write action scripts for both one-time actions and continuous actions.

If actions are used in a script, the action names must be listed in the property.conf file under "scripts".

For a continuous action, such as adding a user, a cookie object persists over the action request. If the continuous action is to add a user, then the corresponding cancel action is to delete a user. The cookie object stores the string.
To define the scripts related to a continuous action, map the script name to the continuous action, then map the cancel script name to the cancel continuous action and specify "is_cancel": true.

```
"name": "cylance_add_user.py",
"actions": [
   "connect_cylance_add_user"
],

"name": "cylance_delete_user.py",
"is_cancel": true,
"actions": [
   "connect_cylance_add_user"
]
```

See Sample Add a User Action Script for Connect and Sample Delete a User Action Script for Connect.

In an action script, you can use the properties object response field to update property values, whether the action succeeded or failed. See Sample Add a User Action Script for Connect.

**Response Objects**

The response objects for action scripts are as follows.

**Mandatory Fields**

"succeeded": boolean to denote if the action succeeded

"troubleshooting" (only mandatory if "succeeded" is false) : string message to display error message if action failed

**Optional Field for Cookie**

"cookie": Optional field for continuous actions to store information that can be used later in the undo script.

**Optional Field for Properties**

"properties": a map/dictionary that contains host properties; the key will be the property name and the value will be the property value.

**Examples for Cookie**

```
response = {"succeeded" : True}
response = {"succeeded" : True, "cookie" : 13452829256}
```

**Examples for Properties**

```
response = {"succeeded" : True,
            "properties":
            "property1": "property_value1",
            "property2": ["value1", "value2", "value3"]
            "property3":
```
Property Resolve Script for Connect

You can write a script that handles host property resolve requests.

If properties are used in a script, the property tags must be listed in the property.conf file under “scripts”.

It is recommended that the property resolve script contain a mapping of the API response fields to the Forescout platform properties, for example:

```
cylance_to_ct_props_map = {
    "state": "connect_cylance_state",
}
```

The property resolve script can also have dependencies defined in the property.conf file. For example, if a MAC address is needed to resolve a property, you can define a dependency on the Forescout platform’s MAC address property.

If a property is not resolved after the Python script is called, the property will be set to irresolvable.

The properties object for property resolve scripts can have scalar, list, composite, or list of composite properties.

See [Sample Resolve Script for Connect](#).

Response Objects

The response objects for property resolve scripts are as follows.

**Mandatory Fields**

"properties": a map/dictionary that contains host properties; the key will be the property name and the value will be the property value.

"error": Error message if script fails for a known reason
Examples
response = {"error" : "Script failed due to server failure."}
response =
{"properties":
    "property1": "property_value1",
    "property2": ["value1", "value2", "value3"]
    "property3":
    {
       "property3_subfield1": "value1",
       "property3_subfield2": "value2"
    }
    "property4": [
       {
          "property4_subfield1": "value1",
          "property4_subfield2": "value2"
       },
       {
          "property5_subfield1": "value1",
           "property5_subfield2": "value2"
       }
    ]
}

Authorization Script for Connect
You can write a script to get one authorization per configuration. Authorizations are stored globally. You can retrieve the authorization from the configuration parameter “connect_authorization_token” whenever you want to use it in scripts. For example:
jwt_token = params["connect_authorization_token"]
An authorization script can leverage input parameters from the user interface, such as a URL.
See Sample Authorization Script for Connect.

Response Objects
The response objects for authorization scripts are as follows.

Mandatory Fields
"token": a string of authorization token. If the authorization is failed, put an empty string "" here.

Token-Based Authorization
There are two approaches to using token-based authorization in scripts.
One approach is to use the authorization feature provided by Connect, which gets the authorization token from the "connect_authorization_token" field whenever you want to use it in scripts. See Authorization Script for Connect.

For a test script, you can check if the token is empty or not. If it is empty, the connection has failed. The sample test script uses this approach. See Sample Test Script for Connect.

The second approach is to get authorization each time a script is called. See the comments in the other sample scripts, for example, see Sample Polling Script for Connect.

**Use App Instance Cache in Connect Scripts**

You can write a script to get non-endpoint data at the per configuration instance of the app. The data is stored as a system description field and is available for that app.

You can retrieve the data from the configuration parameter "connect_app_instance_cache" whenever you want to use it in scripts.

In the script, the field for the response object is "connect_app_instance_cache". Put the value to be saved in this field. If there is any error, put "error" in the response object. For an example, see Sample App Instance Cache Script for Connect.

To use this field value in other scripts, use params.get("connect_app_instance_cache"). For an example, see Sample Add a User Action Script for Connect.

**Use Certificate Validation in Connect Scripts**

If you have defined a checkbox for Validate Server Certificate in the system.conf file, you need to use a built-in SSL object in your Python script.

When using urllib, use the keyword "ssl_context" in the HTTP request. For example:

```python
response = urllib.request.urlopen(request, context=ssl_context)
```

For other examples of "ssl_context", see Sample Connect Script Files.

When using the requests library, use the keyword "ssl_verify" in the HTTPS request. For example:

```python
response = requests.get('https://github.com', verify=ssl_verify)
```

To use the certificate validation feature, upload the entire certificate chain to the Forescout platform.

Certificate validation works with certificates with a Certificate Revocation List (CRL). If the certificates have been uploaded to the Forescout platform, they do not have to be signed by a Certificate Authority (CA).

**Create a Connect App**

To create an app, place files in a zip file.

At a minimum, an app must contain three files:

- system.conf
- property.conf
- one Python script

The zip file of the app can be named for the third-party vendor, for example, cylance.zip.

**Contents of a Zip File**

In every zip file of an app, the system.conf file, property.conf file, and one or more Python scripts must be located at the top level. There can also be two folders.

This sample zip file contains the following:

- property configuration (property.conf or property.json) file (with a .conf or .json suffix). See Define system.conf File.
- system configuration (system.conf or system.json) file (with a .conf or .json suffix). See Define property.conf File.
- Python scripts (each with a .py suffix). See Write Python Scripts for Connect.
- folder for images, for icons in the user interface, such as icons for actions, action groups, property groups, and policy templates. See Define Icons in Connect.
- folder for policies, for policy templates. See Define Policy Templates in Connect.

> The suffixes of the property configuration file and system configuration file do not have to match, for example, an app can have a property.conf and a system.json file.

**File Size Maximum**

The maximum size of the zip file is 10MB. Any single file within the zip file can have a maximum size of 5MB. The maximums are enforced when an app is imported.

**App Folder Paths**

In an app, the files and folders must be as follows:

- All .conf (or .json) and all .py files are under /
- All policy template files are under /policies/nptemplates
Icon files are in the following folders:

- Action group icon files are under /images/np_ng/action_groups
- Action icon files are under /images/np_ng/actions
- Property group icon files are under /images/np_ng/field_groups
- Policy template icon files are under /images/np_ng/templatedirs

**Zip a Connect App Using a Mac**

If you are using the Mac GUI Compress tool to zip the files in an app, you will need to use a workaround.

The Mac GUI Compress tool produces a __MACOSX metadata folder as well as a .DS_store file that are not compliant with the contents of the zip file, which is restricted to the .conf and .py files, and the policies and images folders.

The workaround is to use the Terminal (or command line interface) zip command and not the Mac GUI Compress tool to create the zip file for an app. For example:
```
zip -r dir.zip
```

If you already have a zip file with a __MACOSX metadata folder, you can remove it as follows:
```
zip -d foo.zip __MACOSX .DS_Store
```

If a __MACOSX folder exists in your directory (because you previously used unzip to create it), use the -x option to prevent it from being included in the app.

**Deploy an App with Connect**

This topic describes how to deploy an app with Connect. It is intended for app users and describes downloading and installing an app, licensing, and configuring. Each app has their own configuration details. Refer to the Readme files for each app and also see the following:

- Download a Connect App from GitHub
- Install Connect Plugin
- Connect Add-On Module (Optional)
- Connect User Interface Details

**Download a Connect App from GitHub**

eyeExtend Connect Apps can be found on the Forescout eyeExtend Connect GitHub page located as follows:

https://github.com/Forescout/eyeExtend-Connect

Apps signed by Forescout are in .eca format and are downloaded from GitHub as a .eca image file.
To download a .eca image file for a Connect App from GitHub:

1. Go to the Forescout eyeExtend Connect GitHub page.
2. Select an app, for example, Cylance.

3. Select the .eca image file for that app.
4. Select **Download**.

![Image of Forescout / eyeExtend-Connect](image)

**App Download Issue**

If there is an *Invalid Signature* error when a downloaded app is imported into Connect, the correct procedure in might not have been followed.

For example, it is not correct in Step 3 to right-click on the .eca image file and select **Save link as**. This will result in the *Invalid Signature* error when the app is imported into Connect.

![Image of Forescout / eyeExtend-Connect](image)
Install Connect Plugin

This topic describes how to install the Connect Plugin.

To install the module:

1. Navigate to one of the following Forescout download portals, depending on the licensing mode your deployment is using:
   - Product Updates Portal - *Per-Appliance Licensing Mode*
   - Customer Portal, Downloads Page - *Flexx Licensing Mode*

To identify your licensing mode, select Help > About ForeScout from the Console.

2. Download the module .fpi file.
3. Save the file to the machine where the Console is installed.
4. Log into the Console and select Options from the Tools menu.
5. Select Modules. The Modules pane opens.
6. Select Install. The Open dialog box opens.
7. Browse to and select the saved module .fpi file.
8. Select Install. The Installation screen opens.
9. Select I agree to the License Agreement to confirm that you have read and agree to the terms of the License Agreement and select Install. The installation cannot proceed unless you agree to the license agreement.

   ▶️ The installation begins immediately after selecting Install and cannot be interrupted or canceled.

   ▶️ In modules that contain more than one component, the installation proceeds automatically one component at a time.

10. When the installation completes, select Close to close the window. The installed module is displayed in the Modules pane.

   ▶️ Some components are not automatically started following installation.

Ensure That the Connect Plugin Is Running

After installing the Connect Plugin (and configuring it, if necessary), ensure that it is running.

To verify:

1. Select Tools > Options > Modules.
2. In the Modules pane, hover over the Connect Plugin name to view a tooltip indicating if it is running on Forescout devices in your deployment.

   The name is preceded by one of the following icons:

   - - The Connect Plugin is stopped on all Forescout devices.
   - - The Connect Plugin is stopped on some Forescout devices.
3. If the Connect Plugin is not running, select Start, and then select the relevant Forescout devices.
4. Select OK.

**Connect Add-On Module (Optional)**

The Connect Add-On Module is an optional module that can be installed, which is dependent on the installation of the Connect Plugin. It provides licensing of an additional 20 apps. Two apps are included with the Connect Plugin license, for a total of 22 apps.

If you need to run more than two apps, install the Connect Add-On Module and request a license. Licenses can be for either the Per-Appliance Licensing Mode (PALM) or the Flexx Licensing Mode.

Refer to the *Connect Add-On Module How-to Guide* for more information.

**Without Connect Add-On Module Installed**

If you do not have the Connect Add-On Module installed, you are entitled to two apps with the Connect Plugin license. The Entitlement Status is displayed in the Connect pane.

If you run more than two apps with the Connect Plugin license, the Entitlement Status displays *Out of compliance*.

If an app is not in the Configured state or the Running state, it is not counted as *in use*. The example shows four apps, but only three are both Configured and Running (one app is not Configured), so the count of the number of apps in use is three.

If you stop one app, the number of apps *in use* is two (one app is not Configured and one app is Stopped). The Entitlement Status no longer displays *Out of compliance*. 
With Connect Add-On Module Installed

The Connect Plugin and the Connect Add-On Module are installed under **Options > Modules**.

- The Connect Add-On Module does not require any configuration, nor does it have to be running.

The Entitlement Status is displayed in the Connect pane.
If an app is not in the Configured state or the Running state, it is not counted as in use. The example shows six apps, but only two are both Configured and Running (one is not Configured and three are Stopped), so the count of the number of apps in use is two.

The entitlement status will be in compliance as long as the number of apps does not exceed 22.

**Connect User Interface Details**

This topic provides details of the user interface. See the following:

- [Connect Pane Details](#)
- [System Description Dialog Box Details](#)
- [Configure Policy Templates in Connect](#)
Connect Pane Details

The Connect pane displays existing apps that have been imported and system descriptions that have been configured. There can be multiple apps displayed in this pane.

Columns in Connect Pane

There are several default columns in the Connect pane as follows:

- **Signature**: The signature of the imported app, which has a green checkmark if the signature is valid or an orange caution sign if the signature is not valid or is missing. Apps from Forescout are signed after their creation to ensure their authenticity and integrity. See Import a Signed App.

- **Name**: The name of the third-party integration app, such as Cylance, defined in the system.conf file.

- **Version**: The version of the app, defined in the system.conf file.

- **Author**: The author of the app, defined in the system.conf file.

- **Last Date Modified**: The date that any file in the app was last modified.

- **File Name**: The file name of the app, such as cylance.zip.

- **Import Date**: The date the app was imported.

- **Configured**: The configuration flag, which has a checkmark if the system description is configured. If an app has been imported, but not configured, there will not be a checkmark in this column. To configure the system description, see Add a System Description.

- **Status**: The status of the app. The valid values are Running and Stopped.
At the bottom of the **Connect** pane is the Entitlement Status:

- **Number of apps entitled**: The number of apps to which you are entitled, based on licensing, which can be either 2 or 22. The Connect Plugin license provides two apps. The Connect Add-On Module provides another 20 apps for a total of 22. See [Connect Add-On Module (Optional)](Optional). If there is no valid Connect Plugin license, it will be 0.

- **Number of apps in use**: The number of apps that are Configured and Running. Apps that are not configured and apps that are stopped are not counted as apps in use.

When you right-click on the column titles in the **Connect** pane, a menu for adding and removing columns is displayed.

To add, remove, or reorder the columns on the **Connect** pane, select **Add/Remove Columns**. You can expand the General folder.

Move columns in the lists for **Available Columns** and **Selected Columns** and use the **Move Up** and **Move Down** buttons to reorder the columns in the **Add/Remove Columns** dialog box.

To delete a column, select it and select **Remove Column** in the **Connect** pane.

To select the best fit for a column, right-click a column title and select **Best Fit Column** in the **Connect** pane.
Buttons in Connect Pane

There are several buttons on the Connect pane for apps as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import</td>
<td>Import an App</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit an App</td>
</tr>
<tr>
<td>Update</td>
<td>Update an App</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove an App</td>
</tr>
<tr>
<td>Start</td>
<td>Start an App</td>
</tr>
<tr>
<td>Stop</td>
<td>Stop an App</td>
</tr>
</tbody>
</table>

There are also Apply and Undo buttons on the Connect pane as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Save the Connect configuration to the CounterACT Appliance. See Apply Changes.</td>
</tr>
<tr>
<td>Undo</td>
<td>Undo the previous action performed in Connect.</td>
</tr>
</tbody>
</table>

Import an App

Select the Import button to import apps into Connect in zip or eca format. See also Import a Signed App.
Messages are displayed as the app is imported.

If the app has been imported successfully, a message is displayed at the bottom of the Sending dialog box.

If the app has not been imported successfully, error messages are displayed in the Sending dialog box with a reason, such as an invalid script.

Select Close when the import has finished. If you select Close before the import has finished, it will fail.

An error message is displayed if you try to import a duplicate app with the same name and version as an existing app.
The **System Description** dialog box opens. In this example, there is one column: URL, which has been defined in the system description (in the system.conf file).

![System Description Dialog Box](image)

The maximum number of apps that can be imported is 22.

If a device has not been configured and you select **OK** in the **System Description** dialog box, a warning message is displayed.

To configure a system description, select **Add**. See [Add a System Description](#).

### Import a Signed App

Select the **Import** button to import apps in eca format into Connect. Apps from Forescout are signed after their creation to ensure their authenticity and integrity.

- **Unsigned apps might cause negative impacts on hosting CounterACT Appliances.**

When you import an app, the signature of the app is validated to see if it has a valid Forescout signature. If the validation succeeds, the app is imported.

If the validation fails, an **Invalid Signature** error message is displayed and the app is not imported. See [App Download Issue](#).

To allow an app with an invalid signature to be imported use the following command on the Enterprise Manager:

```
fstool allow_unsigned_connect_app_install true
```

This is a global command. It disables the enforcement of signature validation for all apps that are imported after the command is run, including apps with invalid or missing signatures. The following warning is displayed. Proceed with caution.

![Warning Dialog Box](image)
Update an App

Select the **Update** button to update an app in a limited way.

If the system.conf and property.conf files have changed, you must upgrade the app. See [Upgrade an App](#).

But if you made the following changes, you can use the **Update** button to update an app:

- “version” change only (to a higher or a lower version) in the system.conf file
- Any content change in existing scripts

An error message is displayed if you select an app to update but then select the zip file of another app.

Configured policies are not affected by the update.

Select **Apply** in the Connect pane to complete the update.

Edit an App

Select an existing app in the **Connect** pane and select **Edit** or double-click the existing app to open the system description for it.

![System Description](image)

Remove an App

Remove an existing app before replacing it with an update of the same app.

Select an existing app in the **Connect** pane, select **Remove** to delete it, and confirm the removal.
Before an app is removed, dependencies for properties and actions are checked. An error message is displayed if there are properties or actions configured in a policy when you try to remove the app.

Select Details to view the specific properties and actions that are configured. You may have to remove the policy

Select Close and then select Apply in the Connect pane.

**Start an App**

Select the Start button to start a selected app when it is not in the Running state. Starting an app enables host discovery, property resolves, and actions (if applicable and configured).

When a selected app is running, the Start button is disabled. After an app is started, a status of Running is displayed in the Connect pane.

If an app is not selected, both the Start and Stop buttons are disabled.

If the configuration has not been saved, select Apply and then select Start.

**Stop an App**

Select the Stop button to stop a selected app when it is in the Running state.

For example, if one app has issues, select it and select Stop, and then investigate it. The other apps continue running.
When an app is in the Stopped state, host discovery, property resolves, and actions, (if applicable and configured), are stopped.

When a selected app is not running, the **Stop** button is disabled. After an app is stopped, a status of Stopped is displayed in the **Connect** pane.

If an app is not selected, both the **Start** and **Stop** buttons are disabled.

If the configuration has not been saved, select **Apply** and then select **Stop**.

**Apply Changes**

Select the **Apply** button to save the changes to the configuration.
Select **Close**.

Select **OK**.

**Upgrade an App**

To upgrade an app when the system.conf and property.conf files have changed, you can **Remove** it, **Import** the newer app, and then **Add** the configuration for the system description. See **Remove an App**, **Import an App**, and **Add a System Description**.

Since removing an app removes all configuration, you can also **Export** the system description before removing the app, then **Import** the app and **Import** the system description. See **Export a System Description**, **Import an App**, and **Import a System Description**.

The upgrade steps listed above are needed when the system.conf and property.conf files have changed. If only the content in existing scripts or only the version in the system.conf file have changed, you can update an app. See **Update an App**.

**Menu in Connect Pane**

There is a menu available when you right-click an existing app in the **Connect** pane.

For **Remove**, **Edit**, **Update**, **Import**, **Start**, and **Stop**, see **Buttons in Connect Pane**.

For **Find** and **Export Table**, see **Find Dialog Box** and **Export Table Dialog Box**.
Find Dialog Box
Find a string on the window.

To find a string:
1. Select Find.

2. Enter the text to find, select options or direction, and then select Find.

Export Table Dialog Box
Export the window contents to a spreadsheet.

To export a table:
1. Select Export Table.

2. You can:
   - Export the **Selected rows only** and/or the **Displayed columns only**
   - Change the **File type** to either Comma Separated Values Files (*.csv) or Acrobat Reader Files (*.pdf)
   - Change the folder location using **Browse**

   You cannot enter anything in the **Title field**.
3. Select **OK**. A confirmation is displayed.

   ![Confirmation dialog box](image)

4. Select **Yes**. The table data opens in an Excel spreadsheet. The spreadsheet has a default name based on the date and time, for example, TableData_2020_06_05_09_25.csv-Excel.

   ![Excel spreadsheet](image)

**System Description Dialog Box Details**

After a system description is configured, it is displayed in the **System Description** dialog box.

![System Description dialog box](image)
Columns in System Description Dialog Box

The columns in the **System Description** dialog box are defined in the system.conf file with the “add to column” parameter. When you right-click the column titles, a menu for adding and removing columns is displayed.

To add, remove, or reorder the columns on the **System Description** dialog box, select **Add/Remove Columns**. You can expand the General folder.

Move columns in the lists for **Available Columns** and **Selected Columns** and use the **Move Up** and **Move Down** buttons to reorder the columns in the **Add/Remove Columns** dialog box.

To delete a column, select it and select **Remove Column** in the **System Description** dialog box.

- If there is only one column, you cannot remove it.

To select the best fit for a column, right-click a column title and select **Best Fit Column** in the **System Description** dialog box.
Buttons in System Description Dialog Box

There are several buttons for an integration on the System Description dialog box as follows:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Add a System Description</td>
</tr>
<tr>
<td>Edit</td>
<td>Edit a System Description</td>
</tr>
<tr>
<td>Remove</td>
<td>Remove a System Description</td>
</tr>
<tr>
<td>Test</td>
<td>Test a System Description (Optional)</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refresh App Instance Cache Data</td>
</tr>
<tr>
<td>Import</td>
<td>Import a System Description</td>
</tr>
<tr>
<td>Export</td>
<td>Export a System Description</td>
</tr>
</tbody>
</table>

There are also OK and Cancel buttons on the System Description dialog box:

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Save the system description changes to the CounterACT Appliance.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Cancel the previous action performed in Connect.</td>
</tr>
</tbody>
</table>

Add a System Description

Select Add to display the first configuration panel. The number of panels in the system description are defined in the system.conf file.

The user configuring the system description enters the information on the panel.
Select **Next** to display the next configuration panel that is defined in the system.conf file, such as, the predefined Assign CounterACT Devices panel.

At first, the Assign CounterACT Devices panel has only one option, **Assign all devices by default**, and it is selected so that one device is added.

If you want to add a second device, the Assign CounterACT Devices panel has more options.
The user configuring the system description enters the following information for the predefined fields on the panel:

- **Connecting CounterACT Device**: Select Enterprise Manager or an IP address of the connecting CounterACT device. In an environment where more than one CounterACT device is assigned to a single third-party instance, the connecting CounterACT Appliance functions as a middleman between the third-party instance and the CounterACT Appliance. The connecting CounterACT Appliance forwards all queries and requests to and from the third-party instance.

In general, it is not recommended to use the Enterprise Manager as the connecting CounterACT device. But if you must, make sure that it is not used to discover MAC-only hosts.

- **Assign specific devices**: This CounterACT Appliance is assigned to a third-party instance, but it does not communicate with it directly. All communication between the third-party instance and its assigned CounterACT Appliance is handled by the connecting CounterACT Appliance defined for the third-party instance. All the IP addresses handled by an assigned Appliance must also be handled by the third-party instance to which the Appliance is assigned.
  - Select **Available Devices** and then select an IP address or Appliance name from the Available Devices list.
  - Select **Add**. The selected device will send its requests to the third-party instance through the connecting Appliance.

- **Assign all devices by default**: This is the connecting Appliance to which CounterACT Appliances are assigned by default if they are not explicitly assigned to another connecting Appliance. Select this option to make this connecting Appliance the middleman for all CounterACT Appliances not assigned to another connecting device.

Note the following:

- An error message is displayed if you try to add a device that is already used.
- The focal appliance must be the managing appliance for overlapping IPs.
- If you have apps that discover 50,000 or more endpoints, distribute the apps in such a way so that only up to two of the apps share the same focal (connecting) appliance. An alternative is to split the endpoints across multiple user accounts on multiple servers.
Select **Next** to display the next configuration panel that is defined in the `system.conf` file, such as, the predefined Proxy Server panel.

The user configuring the system description enters the following information for the predefined fields on the panel:

- **Use Proxy Server**: Select this option if your environment routes Internet communications through proxy servers.
- **Proxy Server**: Enter the Fully Qualified Domain Name (FQDN) of the proxy server or the IPv4 address.
- **Proxy Server Port**: Select the port number of the proxy server.
- **Proxy Server Username**: Enter the administrator username used to access the proxy server.
- **Proxy Server Password**: Enter the administrator password used to access the proxy server.
- **Verify Password**: Re-enter the administrator password to verify it.

Select **Next** to display the next configuration panel that is defined in the `system.conf` file, such as, the Cylance Options panel.
The user configuring the system description enters the following information for the predefined fields on the panel:

- **Enable Host Discovery**: Select the checkbox to enable or disable the Discovery Frequency field.

- **Discovery Frequency**: Select a value for the host discovery field. See “host discovery” Field.

- **Number of API queries per unit time**: Select a value for the rate limiter field. See “rate limiter” Field.

Select **Finish** after the last panel.

The configured system description is displayed in the **System Description** dialog box.

You can create multiple system descriptions. To add another system description, select **Add** and repeat **Add a System Description**.
**Edit a System Description**

Select an existing system description in the **System Description** dialog box and select **Edit** to open it. There are tabs for each panel.

Select **OK** to close the dialog box.

**Remove a System Description**

Select an existing system description in the **System Description** dialog box and select **Remove** to remove it. A confirmation is displayed.

Select **More** for details or select **Ok**.
**Scenarios for Remove**

The following table describes different scenarios for removing a system description:

<table>
<thead>
<tr>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The system description being removed has the connecting appliance as the default appliance.</td>
<td>If there is only one system description and it is the default, the remove is allowed.</td>
</tr>
<tr>
<td></td>
<td>If there are two system descriptions, the remove is allowed and the connecting appliance is assigned as the default appliance.</td>
</tr>
<tr>
<td></td>
<td>If there are more than two system descriptions, the remove is not allowed. You must select a new default before removing the system description.</td>
</tr>
</tbody>
</table>

**Test a System Description (Optional)**

To display a Test button in the System Description dialog box, you must have the following defined in the system.conf file:

```
"testEnable":true,
```

Select an app and select Test to call a Python script, for example, to test connectivity of a configuration. The app must be in the Running state.

Also, the app must be saved before selecting Test. Select OK in the System Description pane and then select Apply in the Connect pane to save the system description configuration.

When you select Test, the Sending dialog box opens.

![Sending dialog box](image)

If the connectivity of the system description has been tested successfully, a success message is displayed.

If the test failed, a failure message is displayed with a reason, such as:

- the app is not in Running status
- the script was unable to obtain the JSON authorization token
**Refresh App Instance Cache Data**

To enable the **Refresh** button in the System Description dialog box, you must have the following defined in the system.conf file:

```
"app_instance_cache":true,
```

Select one or more system descriptions in the **System Description** dialog box and select **Refresh** to refresh them.

See “**app_instance_cache**” Field for an example that uses **Refresh**.

**Import a System Description**

Select **Import** to import a saved backup of the configuration. The only supported format is .xml.

**To import a system description:**

1. Select **Import**.
2. To change the folder location, select **Browse**.

![Open dialog box](image)

3. Select **Open**.
4. Select **OK** in the **Import** dialog box.
   a. If the device to be imported contains encrypted fields for passwords, the **Import/Export Password** dialog box opens and prompts you to enter a password with which to encrypt the data.

![Import/Export Password dialog box](image)

   b. Enter the password that you used when you exported the configuration and select **OK**. See **Export a System Description**.
5. If the system description already exists, you have the option to **Skip**, **Duplicate**, or **Overwrite** it.

![Items already exist dialog box](image)
The **Duplicate** button is inactive if there is an "identifier" field set to true in the system.conf file. This is because only one "identifier" is allowed in a system.conf file and creating a duplicate would result in two. If you want to duplicate a system description, you can delete the “identifier” or set the value to false in the system.conf file.

6. The result of the import is displayed.

![Import Result](image)

7. Select **Close**.

**Scenarios for Import**

The following table describes different scenarios for importing a system description:

<table>
<thead>
<tr>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The imported system description is the first configuration of this app.</td>
<td>The import is allowed and the imported focal appliance is set as the default.</td>
</tr>
<tr>
<td>The imported system description will overwrite the default appliance because the user selected <strong>Overwrite</strong> during the import.</td>
<td>The import is allowed and the imported focal appliance is set to the default.</td>
</tr>
<tr>
<td>The imported system description has the default appliance assigned, the default is correct, and is not a duplicate appliance.</td>
<td>If no system description has been configured yet, the import is allowed.</td>
</tr>
<tr>
<td>The imported system description has the default appliance assigned, but the default appliance is not found. For example, the system description might have originated elsewhere.</td>
<td>If more than one system description is configured, the import is allowed and the imported focal appliance is switched to a specific appliance.</td>
</tr>
<tr>
<td></td>
<td>If no system description has been configured yet, the import is allowed and the Enterprise Manager is set as the default appliance. A warning message is displayed.</td>
</tr>
<tr>
<td></td>
<td>If more than one system description is configured and if all other appliances have been assigned to other devices, the import is not allowed.</td>
</tr>
<tr>
<td></td>
<td>If more than one system description is configured and if an appliance is available, one is selected at random.</td>
</tr>
</tbody>
</table>
The imported system description has one or more specific appliances assigned, but the appliances are not found. For example, the system description might have originated elsewhere.

### Result
- If all the specific appliances are not found and if all other appliances have been assigned to other system descriptions, the import is not allowed.
- If all the specific appliances are not found and an appliance is available, one is selected at random.
- If some of the specific appliances are found, the import of the correct specific appliances is allowed.
- If some of the specific appliances are found and if all other appliances have been assigned to other system descriptions, the import is not allowed.

### Export a System Description
Select Export to save a backup of the configuration. The only supported format is .xml.

**To export a system description:**
1. Select Export.
   a. If the devices contain encrypted fields for passwords, the Import/Export Password dialog box opens and prompts you to enter a password with which to encrypt the data.

   ![Import/Export Password dialog box](image)

   - Enter the password.

   ![Import/Export Password dialog box](image)

2. The Export Table dialog box opens. You can select to export the Selected rows only.

   ![Export Table dialog box](image)
3. To change the folder location, select **Browse**.

4. Select **Open**.

5. Select **OK** in the **Export Table** dialog box.

6. Select **Close**.

   *There is no Export for an app.*
Menu in System Description Dialog Box

There is a menu available when you right-click an existing system description in the System Description dialog box.

For Add, Edit, Remove, Test, Refresh, Import, and Export, see Buttons in System Description Dialog Box.

For Find and Export Table, see Find Dialog Box and Export Table Dialog Box. They work the same as in the Connect pane.
Configure Policy Templates in Connect

Policies can be configured from a policy template.

To configure a policy template:

1. In the Forescout Console, select **Policy**.
2. Select **Add** and search for the app name.
3. Expand the folder and select a policy template.
4. Select **Next**.

5. Enter a name for the policy. Optionally, enter a description.

6. Select **Next**. Both the IP Address Range dialog box and the Scope pane open.

7. Use the IP Address Range dialog box to define which endpoints are inspected.

The following options are available:

- **All IPs**: Include all IP addresses in the Internal Network.
- **Segment**: Select a previously defined segment of the network. To specify multiple segments, select **OK** or **Cancel** to close this dialog box, and select **Segments** from the Scope pane.
- **Unknown IP addresses**: Apply the policy to endpoints whose IP addresses are not known. Endpoint detection is based on the endpoint MAC address.
8. Select **OK**. The added range is listed in the Scope pane.

9. Select **Next**.
10. To add a condition, select **Add** in the Condition section and search for the app name to see the properties associated with that app.

11. Configure conditions for the policy and then select **OK**.

12. To add an action, select **Add** in the Actions section and search for the app name to see the actions associated with that app.

13. Configure actions for the policy and then select **OK**.
14. The configured conditions and actions are displayed in the Main Rule.

15. To configure sub-rules, select **Next**.
16. To add a sub-rule, select **Add** and give the sub-rule a name.

17. Configure conditions and actions for the sub-rule and select **OK**.

18. Select **Finish**. The configured policy is displayed in the Policy Manager.
Appendix A: Sample Connect Files

The following sample files are available:

- Sample system.conf File
- Sample property.conf File
- Sample Policy Template .xml File for Connect
- Sample Connect Script Files

Sample system.conf File

```json
{
    "name":"Cylance",
    "version":"1.0.0",
    "author":"Concert Masters",
    "testEnable":true,
    "panels":[
        {
            "title":"Cylance Connection",
            "description":"Cylance Connection",
            "fields":[
                {
                    "display":"URL",
                    "field ID":"connect_cylance_url",
                    "type":"shortString",
                    "mandatory":true,
                    "add to column":true,
                    "show column":false,
                    "identifier":true,
                    "tooltip":"URL"
                },
                {
                    "display":"Tenant ID",
                    "field ID":"connect_cylance_tenant_id",
                    "type":"shortString",
                    "mandatory":true,
                    "add to column":true,
                    "show column":false,
                    "identifier":true,
                    "tooltip":"Tenant ID"
                }
            ]
        }
    ]
}
```


```json
{
    "display":"Application ID",
    "field ID":"connect_cylance_application_id",
    "type":"shortString",
    "mandatory":"true",
    "add to column":"true",
    "show column":"false",
    "tooltip":"Application ID"
},

{
    "display":"Application Secret",
    "field ID":"connect_cylance_application_secret",
    "type":"encrypted",
    "mandatory":"true",
    "tooltip":"Application Secret"
},

{
    "certification validation":true
},

{
    "app_instance_cache":true,
    "display":"Custom configuration refresh interval (in minutes)",
    "min":5,
    "max":2400,
    "value":240
},

{
    "authorization":true,
    "display":"Authorization Interval (in minutes)",
    "min":1,
    "max":100,
    "value":28
}
}
```

"focal appliance":true,
"title":"Assign CounterACT Devices",
"description":"Select the connecting CounterACT device that will communicate with the targeted Cylance instance, including requests by other CounterACT devices. Specific CounterACT devices assigned here cannot be assigned to another server elsewhere. If you do not assign specific devices, by default, all devices will be assigned to one server. This server becomes known as the Default Server."\n
"proxy server":true,
"title":"Proxy Server",
"description":"Select a Proxy Server device to manage all communication between CounterACT and Cylance."

"title":"Cylance Options",
"description":"Cylance Options",
"fields": [\n
{\n  "host discovery": true,\n  "display":"Discovery Frequency",\n  "max":300000,\n  "add to column":"true",\n  "show column":"false",\n  "value":3600\n},\n{\n  "rate limiter": true,\n  "display":"Number of API queries per unit time",\n  "unit":1,\n  "min": 1,\n  "max":1000,\n  "add to column":"true",\n  "show column":"false",\n  "value":100\n}\n]
Sample property.conf File

{
    "name": "Cylance",
    "groups": [
        {
            "name": "connect_cylance_cylance",
            "label": "Cylance"
        }
    ],
    "properties": [
        {
            "tag": "connect_cylance_state",
            "label": "Cylance State",
            "description": "Cylance State",
            "type": "string",
            "options": [
                {
                    "name": "Online",
                    "label": "Online"
                },
                {
                    "name": "Offline",
                    "label": "Offline"
                }
            ],
            "group": "connect_cylance_cylance",
            "resolvable": true,
            "require_host_access": false,
            "inventory": {
                "enable": true,
                "description": "Inventory of Cylance State"
            },
            "asset_portal": true,
            "track_change": {
                "enable": true,
                "label": "Cylance State Changed",
                "description": "Track Change property for cylance state"
            }
        }
    ]
}
"dependencies": [
  {
    "name": "mac",
    "redo_new": true,
    "redo_change": true
  }
],
{
  "tag": "connect_cylance_last_logged_in_user",
  "label": "Cylance Last Logged In User",
  "description": "Cylance Last Logged In User",
  "type": "string",
  "group": "connect_cylance_cylance",
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
},
{
  "tag": "connect_cylance_mac_addresses",
  "label": "Cylance Mac Addresses",
  "description": "Cylance Mac Addresses",
  "type": "string",
  "group": "connect_cylance_cylance",
  "list": true,
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
},
}
"tag": "connect_cylance_ip_addresses",
"label": "Cylance IP Addresses",
"description": "Cylance IP Addresses",
"type": "string",
"group": "connect_cylance_cylance",
"list": true,
"overwrite": true,
"dependencies": [
  {
    "name": "mac",
    "redo_new": true,
    "redo_change": true
  }
]
},
{
  "tag": "connect_cylance_is_safe",
  "label": "Cylance is Safe",
  "description": "Cylance is Safe",
  "type": "boolean",
  "group": "connect_cylance_cylance",
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
},
{
  "tag": "connect_cylance_id",
  "label": "Cylance ID",
  "description": "Cylance ID",
  "type": "string",
  "group": "connect_cylance_cylance",
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
}
"redo_new": true,
"redo_change": true
}
]
],
{
"tag": "connect_cylance_policy",
"label": "Cylance Policy",
"description": "Cylance Policy",
"type": "composite",
"group": "connect_cylance_cylance",
"inventory": {
"enable": true,
"description": "Inventory of Cylance Policy"
},
"subfields": [
{
"tag": "id",
"label": "ID",
"description": "Policy ID",
"type": "string",
"inventory": true
},
{
"tag": "name",
"label": "Name",
"description": "Policy Name",
"type": "string",
"inventory": true
}
],
"dependencies": [
{
"name": "mac",
"redo_new": true,
"redo_change": true
}
]


```json
{
  "tag": "connect_cylance_add_user_action",
  "label": "Cylance Add User Action Status",
  "description": "Cylance add user action status",
  "type": "composite",
  "resolvable": false,
  "group": "connect_cylance_cylance",
  "subfields": [
    {
      "tag": "status",
      "label": "Status",
      "description": "Action Status - succeeded, failed or canceled",
      "type": "string"
    },
    {
      "tag": "time",
      "label": "Complete Time",
      "description": "completed time",
      "type": "date"
    }
  ],
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
},

"action_groups": [
  {
    "name": "connect_cylance_cylance",
    "label": "Cylance"
  }
],

"actions": [
```
{
    "name": "connect_cylance_add_user",
    "label": "Add User",
    "group": "connect_cylance_cylance",
    "description": "Add New User",
    "ip_required": false,
    "threshold_percentage": 1,
    "params": [
        {
            "name": "cylance_email",
            "label": "Email address",
            "description": "Cylance email address",
            "type": "string"
        },
        {
            "name": "cylance_first_name",
            "label": "First name",
            "description": "Cylance first name",
            "type": "string"
        },
        {
            "name": "cylance_last_name",
            "label": "Last name",
            "description": "Cylance last name",
            "type": "string"
        }
    ],
    "dependencies": [
        {
            "name": "mac",
            "redo_new": true,
            "redo_change": true
        }
    ],
    "undo": {
        "label": "Cancel Cylance Add User",
        "description": "Remove Added User"
    }
}


{  
    "name": "cylance_resolve.py",
    "properties": [
        "connect_cylance_state",
        "connect_cylance_last_logged_in_user",
        "connect_cylance_mac_addresses",
        "connect_cylance_is_safe",
        "connect_cylance_id"
    ]
},

{  
    "name": "cylance_add_user.py",
    "actions": [
        "connect_cylance_add_user"
    ]
},

{  
    "name": "cylance_delete_user.py",
    "is_cancel": true,
    "actions": [
        "connect_cylance_add_user"
    ]
},

{  
    "name": "cylance_test.py",
    "test": true
},

{  
    "name": "cylance_poll.py",
    "discovery": true
},

{  
    "name": "cylance_authorization.py",
    "authorization": true
},

}
Sample Policy Template .xml File for Connect

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<RULES>

  <RULE APP_VERSION="8.2.0-260" CACHE_TTL="259200"
    CACHE_TTL_SYNCED="true" CLASSIFICATION="REG_STATUS" DESCRIPTION=""
```
<GROUP_IN_FILTER/>

<INACTIVITY_TTL TTL="0" USE_DEFAULT="true"/>

<ADMISSION_RESOLVE_DELAY TTL="0" USE_DEFAULT="true"/>

<MATCH_TIMING RATE="28800" SKIP_INACTIVE="true">
  <ADMISSION ALL="true"/>
</MATCH_TIMING>

<EXPRESSION EXPR_TYPE="SIMPLE">
  <!--Rule expression. Rule name is: Cylance Compliance-->
  <CONDITION EMPTY_LIST_VALUE="false" FIELD_NAME="connect_cylance_id" LABEL="Cylance ID" LEFT_PARENTHESIS="0" LOGIC="AND" PLUGIN_NAME="Integration connect" PLUGIN_UNIQUE_NAME="connect" PLUGIN_VERSION="1.0.0" PLUGIN_VERSION_NUMBER="24" RET_VALUE_ON_UNKNOWN="IRRESOLVED" RIGHT_PARENTHESIS="0">
    <FILTER CASE_SENSITIVE="false" FILTER_ID="-7121200522400372370" TYPE="any">
      <VALUE VALUE2="Any"/>
    </FILTER>
  </CONDITION>
</EXPRESSION>

<EXCEPTION NAME="ip" UNKNOWN_EVAL="UNMATCH"/>

<EXCEPTION NAME="mac" UNKNOWN_EVAL="UNMATCH"/>

<EXCEPTION NAME="nbthost" UNKNOWN_EVAL="UNMATCH"/>

<EXCEPTION NAME="user" UNKNOWN_EVAL="UNMATCH"/>

<EXCEPTION NAME="group" UNKNOWN_EVAL="UNMATCH"/>

<ORIGIN NAME="CUSTOM"/>

<UNMATCH_TIMING RATE="28800" SKIP_INACTIVE="true">
  <ADMISSION ALL="true"/>
</UNMATCH_TIMING>

<SEGMENT ID="-838284172664142831" NAME="CABAB-network-vlan4">
  <RANGE FROM="10.100.4.0" TO="10.100.4.255"/>
</SEGMENT>

<RULE_CHAIN>
  <INNER_RULE APP_VERSION="8.2.0-260" CACHE_TTL="259200" CACHE_TTL_SYNCED="true" CLASSIFICATION="REG STATUS" DESCRIPTION=""/>
Sample Connect Script Files

The following sample Python scripts, with commenting, are available:

- Sample Test Script for Connect
- Sample Polling Script for Connect
- Sample Resolve Script for Connect
- Sample App Instance Cache Script for Connect
- Sample Add a User Action Script for Connect
- Sample Delete a User Action Script for Connect
- Sample Authorization Script for Connect
Sample Test Script for Connect

```python
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import json
import urllib.request
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta

# CONFIGURATION
# All server configuration fields will be available in the 'params' dictionary.
jwt_token = params['connect_authorization_token']  # auth token

response = {}

# Like the action response, the response object must have a "succeeded" field to denote success. It can also optionally have
# a "result_msg" field to display a custom test result message.
if "connect_authorization_token" in params and params['connect_authorization_token'] != "":
    response["succeeded"] = True
    response["result_msg"] = "Successfully connected."
else:
    response["succeeded"] = False
    response["result_msg"] = "Could not connect to Cylance server."
```

```
Sample Polling Script for Connect

```python
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta

# Mapping between Cylance API response fields to CounterACT properties
cylance_to_ct_props_map = {
    "state": "connect_cylance_state",
    "mac_addresses": "connect_cylance_mac_addresses",
    "id": "connect_cylance_id"
}

# CONFIGURATION
url = params["connect_cylance_url"]  # Server URL
response = {}
endpoints=[]

# Check if we have valid auth token or not before processing.
if "connect_authorization_token" in params and
params["connect_authorization_token"] != "":
    # ***** PART 2 - QUERY FOR DEVICES ***** #
jwt_token = params["connect_authorization_token"]
GETMAC_URL = url + "/devices/v2/"

    device_headers = {
"Content-Type": "application/json; charset=utf-8",
"Authorization": "Bearer " + str(jwt_token)}

    # Get MAC data
    request = urllib.request.Request(GETMAC_URL, headers=device_headers)
    try:
        r = urllib.request.urlopen(request, context=ssl_context)
        request_response = json.loads(r.read())

        # For polling, the response dictionary must contain a list
called "endpoints", which will contain new endpoint information. Each
endpoint
        # must have a field named either "mac" or "ip". The endpoint
object/dictionary may also have a "properties" field, which contains
property information in the format
```
# {"propert_name": "property_value"}. The full response object, for example would be:

# {"endpoints":
#    [
#      {"mac": "001122334455",
#       "properties":
#          {"property1": "property_value", "property2": "property_value2"}
#      }
#    ]
#}

for endpoint_data in request_response["page_items"]:
    endpoint = {}
    mac_with_dash = endpoint_data["mac_addresses"][0]
    mac = ".".join(mac_with_dash.split("-"))
    endpoint["mac"] = mac
    properties = {}
    for key, value in endpoint_data.items():
        if key in cylance_to_ct_props_map and key is not "mac_addresses":
            properties[cylance_to_ct_props_map[key]] = value

    endpoint["properties"] = properties
    endpoints.append(endpoint)

response["endpoints"] = endpoints

except:
    response["error"] = "Could not retrieve endpoints."

else:
    response["error"] = "Unauthorized"

**Sample Resolve Script for Connect**

```
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta

# Mapping between Cylance API response fields to CounterACT properties
cylance_to_ct_props_map = {
```
"state": "connect_cylance_state",
"last_logged_in_user": "connect_cylance_last_logged_in_user",
"mac_addresses": "connect_cylance_mac_addresses",
"is_safe": "connect_cylance_is_safe",
"id": "connect_cylance_id"
}

# CONFIGURATION
# All server configuration fields will be available in the 'params' dictionary.
url = params["connect_cylance_url"] # Server URL

response = {}  # Check if we have valid auth token or not before processing.
if "connect_authorization_token" in params and
params["connect_authorization_token"] != ":
    # For properties and actions defined in the 'property.conf' file, CounterACT properties can be added as dependencies. These values will be
    # found in the params dictionary if CounterACT was able to resolve
    # the properties. If not, they will not be found in the params dictionary.
jwt_token = params["connect_authorization_token"]
if "mac" in params:
    mac = '-'.join(params["mac"][:i:i+2] for i in range(0,12,2))
    GETMAC_URL = url + "/devices/v2/macaddress/" + mac
    device_headers = {"Content-Type": "application/json; charset=utf-8", "Authorization": "Bearer " + str(jwt_token)}

    # Get MAC data
    request = urllib.request.Request(GETMAC_URL, headers=device_headers)
    try:
        r = urllib.request.urlopen(request, context=ssl_context)
        request_response = json.loads(r.read())

        # All responses from scripts must contain the JSON object 'response'. Host property resolve scripts will need to populate a
        # 'properties' JSON object within the JSON object 'response'. The 'properties' object will be a key, value mapping between the
        # CounterACT property name and the value of the
        properties = {}
if request_response:
    return_values = request_response[0]
    for key, value in return_values.items():
        if key in cylance_to_ct_props_map:
            properties[cylance_to_ct_props_map[key]] = value

        response['properties'] = properties
    except Exception as e:
        response['error'] = "Could not resolve properties: {}".format(str(e))
    else:
        response['error'] = "No mac address to query the endpoint for."
    else:
        response['error'] = "Unauthorized"

Sample App Instance Cache Script for Connect

import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta
from urllib.request import HTTPError, URLError

# CONFIGURATION
url = params['connect_cylance_url']  # Server URL

response = {}
# Check if we have valid auth token or not before processing.
if "connect_authorization_token" in params and
    params['connect_authorization_token'] != "":
    # ***** PART 2 - QUERY FOR USERS ***** #
    jwt_token = params['connect_authorization_token']
    try:
        GETUSERS_URL = url + "/users/v2/"
        device_headers = {'Content-Type': "application/json; charset=utf-8", "Authorization": "Bearer " + str(jwt_token)}

        # Get users to save as app instance cache
request = urllib.request.Request(GETUSERS_URL, headers=device_headers)

r = urllib.request.urlopen(request, context=ssl_context)
request_response = json.loads(r.read())
response_obj = {}

for user_data in request_response['page_items']:
    user = {}
    email = user_data['email']
    user['id'] = user_data['id'];
    user['first_name'] = user_data['first_name']
    user['last_name'] = user_data['last_name']
    response_obj[email] = user

    # For app instance cache, use the 'connect_app_instance_cache' to be the response key.
    # The value needs to be a string. It can be a json string containing different fields or any other format,
    # depending on how you want to use the data in other scripts.
    response['connect_app_instance_cache'] = json.dumps(response_obj)
    logging.debug("response: ".format(response))

except HTTPError as e:
    response['error'] = "Could not connect to Cylance. Response code: {}".format(e.code)
except URLError as e:
    response['error'] = "Could not connect to Cylance. {}".format(e.reason)
except Exception as e:
    response['error'] = "Could not connect to Cylance. {}".format(str(e))

else:
    # In the response, put 'error' to indicate the error message.
    # 'connect_app_instance_cache' is optional when it has error.
    # if connect_app_instance_cache is in the response object, it will overwrite previous cache value.
    # Otherwise, the previous cache value will remain the same.
    response['connect_app_instance_cache'] = "{}
    response['error'] = "No authorization token found"
Sample Add a User Action Script for Connect

```python
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta
from urllib.request import HTTPError, URLError

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params['connect_cylance_url']  # Server URL

response = {}
properties = {}
action_status = {}

# Check if we have valid auth token or not before processing.
if 'connect_authorization_token' in params and
params['connect_authorization_token'] != '':
    # ***** Execute add user action *****#
    jwt_token = params['connect_authorization_token']
    # connect_app_instance_cache data is available when you enable
    app_instance_cache feature in system.conf
    # the data is available in the 'params' dictionary.
    # In this example, we are getting the existing users.
    user_list = params.get('connect_app_instance_cache')
    user_email = params['cylance_email']
    if user_list is not None and user_email in user_list:
        # return action failed if user already exists
        logging.debug("User {} already exists. ".format(user_email))
        response['succeeded'] = False
        response['troubleshooting'] = "User already exists."
        action_status['status'] = "Failed. User already exists."
        action_status['time'] = int(time.time())
        # Add properties dictionary to response to resolve properties.
        It is optional
        properties['connect_cylance_add_user_action'] = action_status
        response['properties'] = properties
    else:
```

versions 1.4.0 and 1.4.1 124
ADD_USER_URL = url + "/users/v2/"

device_headers = {"Content-Type": "application/json; charset=utf-8", "Authorization": "Bearer " + str(jwt_token)}

body = dict()

# For actions, you can specify user inputted parameters that must be defined in the 'property.conf' file. These parameters will take in user input
# from the CounterACT console and will be available in the 'params' dictionary.

body["email"] = params["cylance_email"]
body["user_role"] = "00000000-0000-0000-0000-000000000001"
body["first_name"] = params["cylance_first_name"]
body["last_name"] = params["cylance_last_name"]

zones = dict()
zone_array = list()
zones["id"] = "0927bf62-83f4-4766-a825-0b5d2e9749d0"
zones["role_type"] = "00000000-0000-0000-0000-000000000002"
zones["role_name"] = "User"
zone_array.append(zones)
body["zones"] = zone_array

json_body = json.dumps(body).encode('utf-8')

request = urllib.request.Request(ADD_USER_URL, headers=device_headers, data=json_body)

try:
    r = urllib.request.urlopen(request, context=ssl_context)
    # For actions, the response object must have a field named "succeeded" to denote if the action succeeded or not.
    # The field "troubleshooting" is optional to display user defined messages in CounterACT for actions. The field
    # "cookie" is available for continuous/cancellable actions to store information for the same action. For this example,
    # the cookie stores the id of the user, which will be used to delete the same user when this action is cancelled.
    response["succeeded"] = True
    request_response = json.loads(r.read())
    id = request_response['id']
    logging.debug("The cookie content is ".format(id))
    response["cookie"] = id
    action_status["status"] = "Succeeded"
action_status["time"] = int(time.time())
properties["connect_cylance_add_user_action"] = action_status
properties["connect_cylance_last_logged_in_user"] = params["cylance_email"]
response["properties"] = properties

except HTTPError as e:
    response["succeeded"] = False
    response["troubleshooting"] = "Failed action. Response code: {}".format(e.code)
    action_status["status"] = "Failed. HTTPError."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status

except URLError as e:
    response["troubleshooting"] = "Failed action. {}".format(e.reason)
    response["succeeded"] = False
    action_status["status"] = "Failed. URLError."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status

except Exception as e:
    response["troubleshooting"] = "Failed action. {}".format(str(e))
    response["succeeded"] = False
    action_status["status"] = "Failed. Exception."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status

else:
    response["succeeded"] = False
    response["troubleshooting"] = "Unauthorized"
    action_status["status"] = "Failed. Unauthorized."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status
    response["properties"] = properties
Sample Delete a User Action Script for Connect

```python
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta
from urllib.request import HTTPError, URLError

# CONFIGURATION
# All server configuration fields will be available in the 'params' dictionary.
url = params['connect_cylance_url']  # Server URL

response = {}
properties = {}
action_status = {}

# Check if we have valid auth token or not before processing.
if "connect_authorization_token" in params and params['connect_authorization_token'] != "":
    # ***** PART 2 - DELETE USER ***** #
    # Here, the cookie that was set in adding the user is being used. The user id is used to delete the user.
    jwt_token = params['connect_authorization_token']
    DELETE_USER_URL = url + '/users/v2/' + params['cookie']
    device_headers = {'Authorization': 'Bearer ' + str(jwt_token)}
    request = urllib.request.Request(DELETE_USER_URL, headers=device_headers, method='DELETE')
    try:
        r = urllib.request.urlopen(request, context=ssl_context)
        response['succeeded'] = True
        action_status['status'] = "Succeeded"
        action_status['time'] = int(time.time())
        properties['connect_cylance_add_user_action'] = action_status
        response['properties'] = properties
    except HTTPError as e:
        response['troubleshooting'] = "Failed action. Response code: {}".format(e.code)
        response['succeeded'] = False
        action_status['status'] = "Failed. HTTPError."
        action_status['time'] = int(time.time())
```

Versions 1.4.0 and 1.4.1
properties["connect_cylance_add_user_action"] = action_status
response["properties"] = properties

except URLError as e:
    response["troubleshooting"] = "Failed action.\n{}\n".format(e.reason)
    response["succeeded"] = False
    action_status["status"] = "Failed. URLError."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status
    response["properties"] = properties

except Exception as e:
    response["troubleshooting"] = "Failed action.\n{}\n".format(str(e))
    response["succeeded"] = False
    action_status["status"] = "Failed. Exception."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status
    response["properties"] = properties

else:
    response["succeeded"] = False
    response["troubleshooting"] = "Unauthorized"
    action_status["status"] = "Failed. Unauthorized."
    action_status["time"] = int(time.time())
    properties["connect_cylance_add_user_action"] = action_status
    response["properties"] = properties

Sample Authorization Script for Connect

import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
import json
import urllib.request
import time
from time import gmtime, strftime, sleep
from datetime import datetime, timedelta

# CONFIGURATION
# All server configuration fields will be available in the 'params' dictionary.
url = params["connect_cylance_url"]  # Server URL
tenant = params["connect_cylance_tenant_id"]  # Tenant ID
app = params["connect_cylance_application_id"]  # Application ID
secret = params["connect_cylance_application_secret"]  # Application Secret

# ***** START - AUTH API CONFIGURATION ***** #
timeout = 1800  # 30 minutes from now
now = datetime.utcnow()
timeout_datetime = now + timedelta(seconds=timeout)
epoch_time = int((now - datetime(1970, 1, 1)).total_seconds())
epoch_timeout = int((timeout_datetime - datetime(1970, 1, 1)).total_seconds())
jti_val = str(uuid.uuid4())
claims = {
    "exp": epoch_timeout,
    "iat": epoch_time,
    "iss": "http://cylance.com",
    "sub": app,
    "tid": tenant,
    "jti": jti_val,
}
encoded = jwt.encode(claims, secret, algorithm='HS256')
payload = {"auth_token": encoded.decode("utf-8")}
headers = {"Content-Type": "application/json; charset=utf-8"}

# Making an API call to get the JWT token
request = urllib.request.Request(url + "/auth/v2/token", headers=headers, data=bytes(json.dumps(payload), encoding="utf-8"))
response = {}
try:
    # To use the server validation feature, use the keyword 'ssl_context' in the http request
    resp = urllib.request.urlopen(request, context=ssl_context)
    jwt_token = json.loads(resp.read())['access_token']  # access_token to be passed to GET request

    response["token"] = jwt_token
except:
    response["token"] = ""