FORESCOUT

Forescout

eyeExtend Connect Module: Connect Plugin Application Building and Deployment Guide v2.0.3

Contact Information

Forescout Technologies, Inc.
2400 Dallas Pkwy, Suite 230, Plano, TX 75093
https://www.forescout.com/support-hub/
Toll-Free (US): 1-866-377-8773
Tel (Intl): 1-708-237-6591

About the Documentation

- Refer to the Documentation Portal for additional technical documentation: https://docs.forescout.com/
- Have feedback or questions? Write to us at documentation@forescout.com

Legal Notice

PDF Created: February 25, 2025

Table of

About the Connect plugin

This topic provides a general overview of the Connect plugin, which is part of the eyeExtend Connect Module.

The Connect plugin provides an infrastructure for integrating third-party vendors with the Forescout Platform. .

Supported Forescout Platform version

The following table lists the Forescout Platform version that works with each version covered by this guide:

Plugin version	Delivered with Connect module version	Works with Forescout Platform version
2.0.3	2.1.3	8.3 and above

About this guide

This guide aims to instruct users on how to build and deploy applications with the Connect plugin. The guide is divided in 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.

Connect module target 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.

Connect module architecture

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

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 process has to start to run 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.

To improve overall security and prevent potential impact on other Forescout Platform appliances, Connect apps are now limited in the amount of CPU and memory they can use.

Connect module 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, user name, 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.

Note:
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 live in a zip file, which gets imported into Connect.

Note:
In this guide, the examples refer to an app called SampleApp.

What's an app builder?

An app builder does the following:

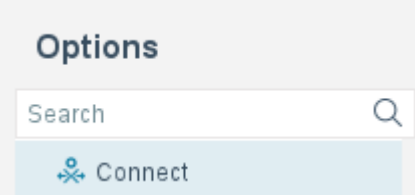
- *Defines* content in configuration files for the system description, which is the connection to the third-party vendor, as well as the properties, actions, and policy templates for the integration
- *Writes* Python scripts to accomplish tasks in the Forescout Platform such as resolve properties take 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.

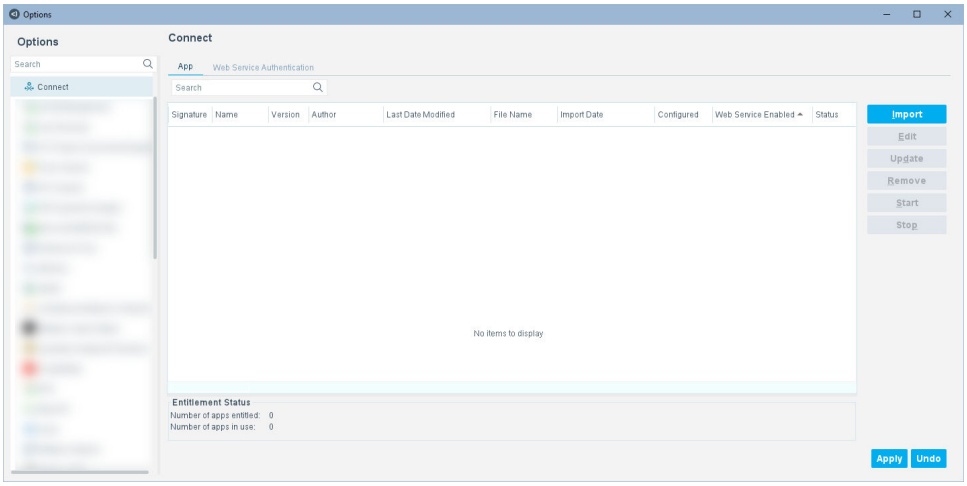
Connect user interface overview

This topic provides an overview of the Connect user interface and related steps.

After installation, you'll see Connect displayed under Options:



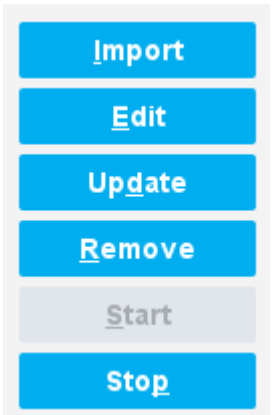
The default is for the Connect pane to be blank. No apps have been imported yet, and no system descriptions have been configured:



There are two tabs on the Connect pane:

1. Use the App tab to manage apps
2. Use the Web Service Authentication tab to configure authentication for the Connect web service

There are several buttons on the Connect pane:

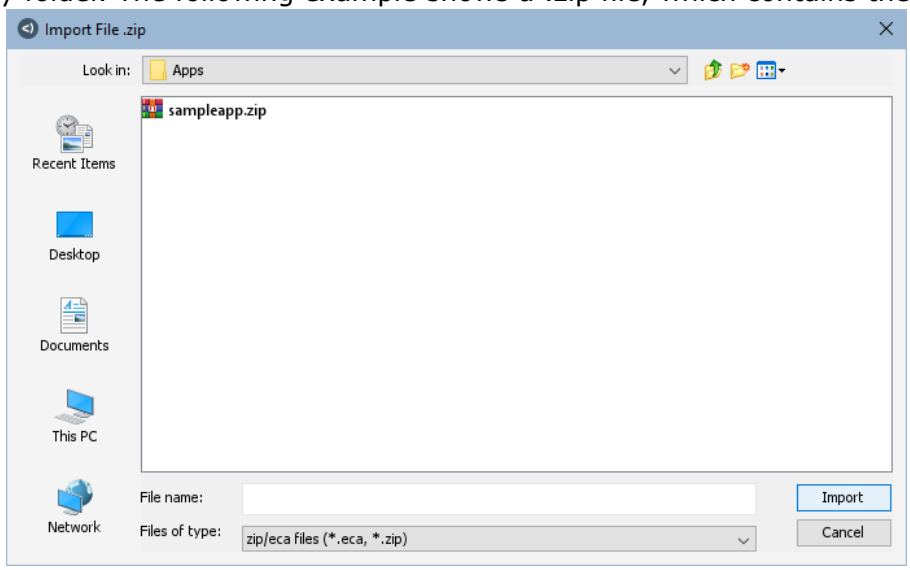


The buttons on the Connect pane are as follows:

Button	Description
Import	Import an app
Edit	Edit an app

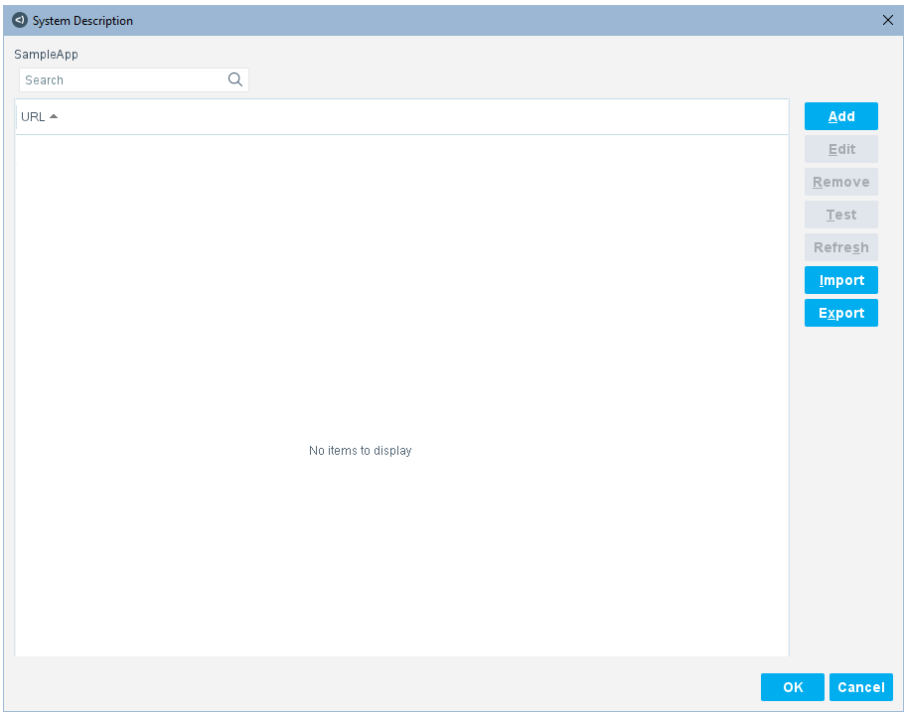
Button	Description
Update	Update an app
Remove	Remove an app
Start	Start an app
Stop	Stop an app

1. To import apps into Connect, select Import. Apps are in zip or eca format and 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.

2. Once imported, the System Description box opens:



Note:
Third-party vendor integrations are displayed inside the Connect pane, not under Options.

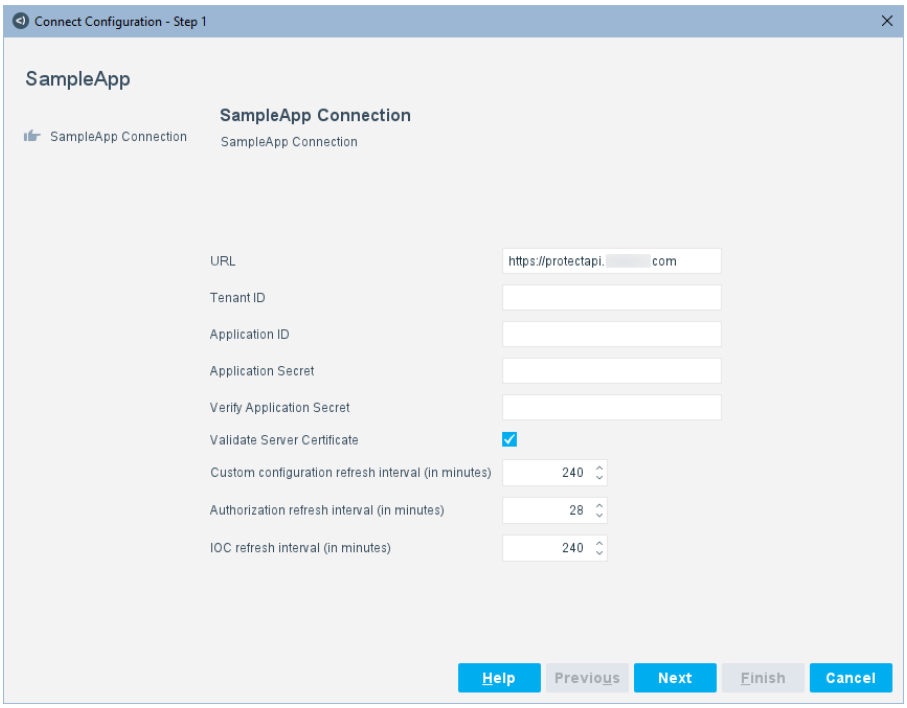
3. Select Apply to save the configuration
4. Enable the Start button to start the app. To stop the app, select Stop.
5. You can then choose an existing app to open by selecting Edit. The System Description dialog box will open. If no system descriptions have been configured, the System Description dialog box will be blank:

There are several buttons for a system description as follows:

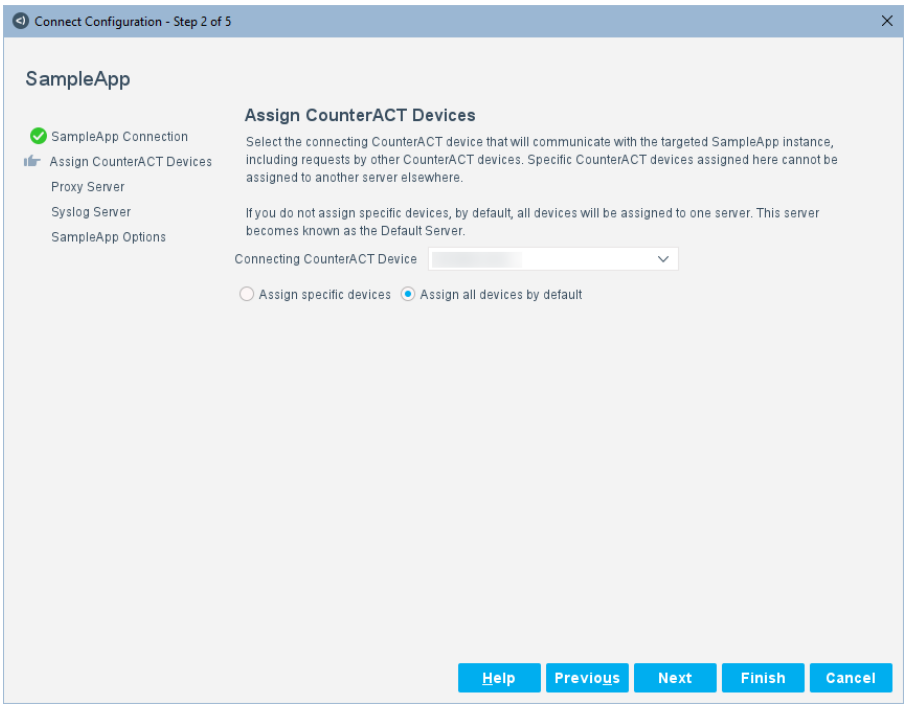
The buttons on the System Description dialog box are as follows:

Button	Description
Add	Add a system description
Edit	Edit a system description
Remove	Remove a system description
Test	(Optional) Test a system description
Refresh	Refresh app feature
Import	Import a system description
Export	Export a system description

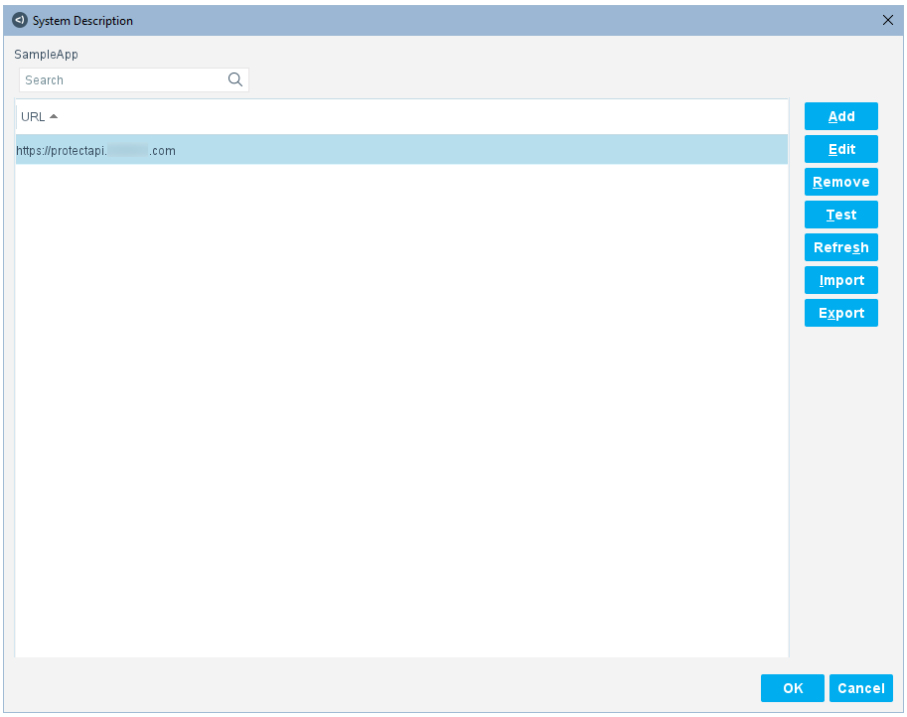
6. Select Add on the System Description dialog box to add a system description, which defines a connection to a third-party vendor. The system.conf file determines the configuration panels and the fields on each panel that can be configured when you select Add:



7. Select Next to display the next configuration panel that is defined in the system.conf file:



8. Select Next to display the next configuration panel that is defined in the system.conf file. There can be multiple panels.
9. Select Finish - the configuration 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.

This topic describes how to create an app. Users can continue on to see the following topics:

- Define system.conf file
- Define property.conf file
- Create policy template XML file for Connect
- Write python scripts for Connect
- Use the Connect web service
- Create a Connect app

Create a Connect app

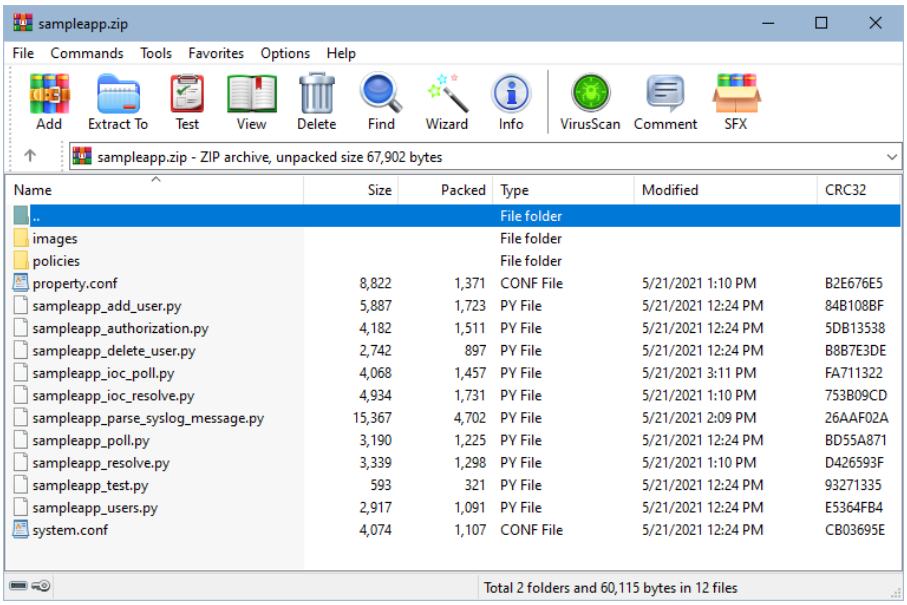
To create an app, you must first place files into a zip file.

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. In this guide, the sample app is called sampleapp.

1. 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. You'll also see 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.
- A 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.
- A folder for policies, for policy templates - see Define Policy Templates in Connect.

  Note:
  The suffixes of the property configuration file and system configuration file don't have to match. For example, an app can have both a property.conf and a system.json file.

2. Fill 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.
3. To submit a Connect app to Forescout for review, use the following email alias at forescout.com: connect-app-submission
4. Format the App Folder paths:

   In the 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
5. Zip a Connect app using MAC:

If you're 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 meta data 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 meta data 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.

Define system.conf file

This topic describes how to define the 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 contains the following:

- App information, such as name, version, and author, as well as specify certain features. 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.

Below is a sample system.conf file (split into two parts):

- Five panels have been defined
- The first panel has been defined with eight fields

```
{
   "name":"SampleApp",
   "version":"1.1.1",
   "author":"Concert Masters",
   "testEnable":true,
   "web service":true,
   "focal only syslog": true,
   "panels":[
      {
         "title":"SampleApp Connection",
         "description":"SampleApp Connection",
         "fields":[
            {
               "display":"URL",
               "field ID":"connect_sampleapp_url",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"true",
               "identifier":"true",
               "tooltip":"URL"
            },
            {
               "display":"Tenant ID",
               "field ID":"connect_sampleapp_tenant_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Tenant ID"
            },
            {
               "display":"Application ID",
               "field ID":"connect_sampleapp_application_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Application ID"
            },
            {
               "display":"Application Secret",
               "field ID":"connect_sampleapp_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 refresh interval (in minutes)",
              "min":1,
              "max":100,
              "value":28
            },
            {
               "ioc_poll":true,
               "display":"IOC refresh interval (in minutes)",
               "min":5,
               "max":2400,
               "value":240
            }
         ]
      },
```

App name, version, author

Specify features

First panel

First field on first panel

Second field on first panel

Third field on first panel

Fourth field on first panel

Fifth field on first panel

Sixth field on first panel

Seventh field on first panel

Eighth field on first panel

```
{
   "focal appliance":true,
   "title":"Assign CounterACT Devices",
   "description":"<html>Select the connecting CounterACT device that will communicate with the targeted
   SampleApp instance, including requests by other CounterACT devices. Specific CounterACT devices assigned
   here cannot be assigned to another server elsewhere.<br><br>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.<html>"
},
{
   "proxy server":true,
   "title":"Proxy Server",
   "description":"<html>Select a Proxy server to manage all communication between CounterACT and
   SampleApp.</html>"
},
{
   "syslog source":true,
   "title":"Syslog Source",
   "description":"<html>Define a Syslog source that sends syslog message to Forescout.</html>"
},
{
   "title":"SampleApp Options",
   "description":"SampleApp Options",
   "fields": [
     {
        "host discovery": true,
        "display":"Discovery Frequency",
        "max":300000,
        "add to column":"true",
        "show column":"false",
        "value":3600
     },
     {
        "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
     }
   ]
}
]
}
```

Second panel

Third panel

Fourth panel

Fifth panel

First field on fifth panel

Second field on fifth panel

Define name, version, and author in system.conf

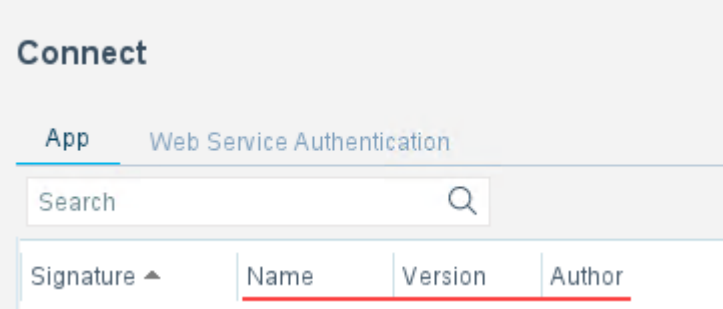
This topic describes how to define the fields at the top of the system.conf file.

The “name”, “version”, and “author” fields in the system.conf file are all required. Users will see an optional, predefined Test button and fields to enable the Connect web service, used to specify if only the focal appliance will receive the syslog message:

```
{
  "name":"appname",
  "version":"1.0.0",
  "author":"Firstname Lastname",
  "testEnable":true,
  "web service":true,
  "focal only syslog": true,}
```

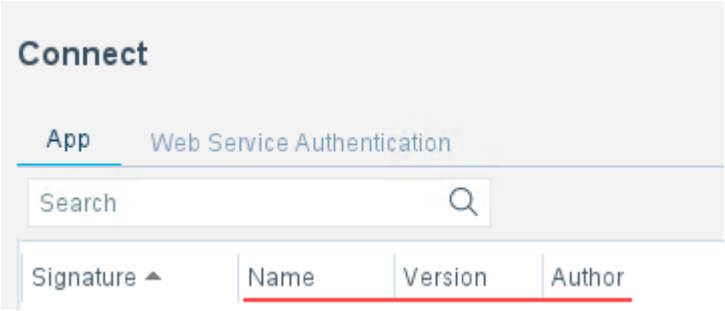
Name, version, and author in user interface

The name, version, and author defined, in the system.conf file, are displayed in the user interface on the Connect pane.



Fields in the user interface

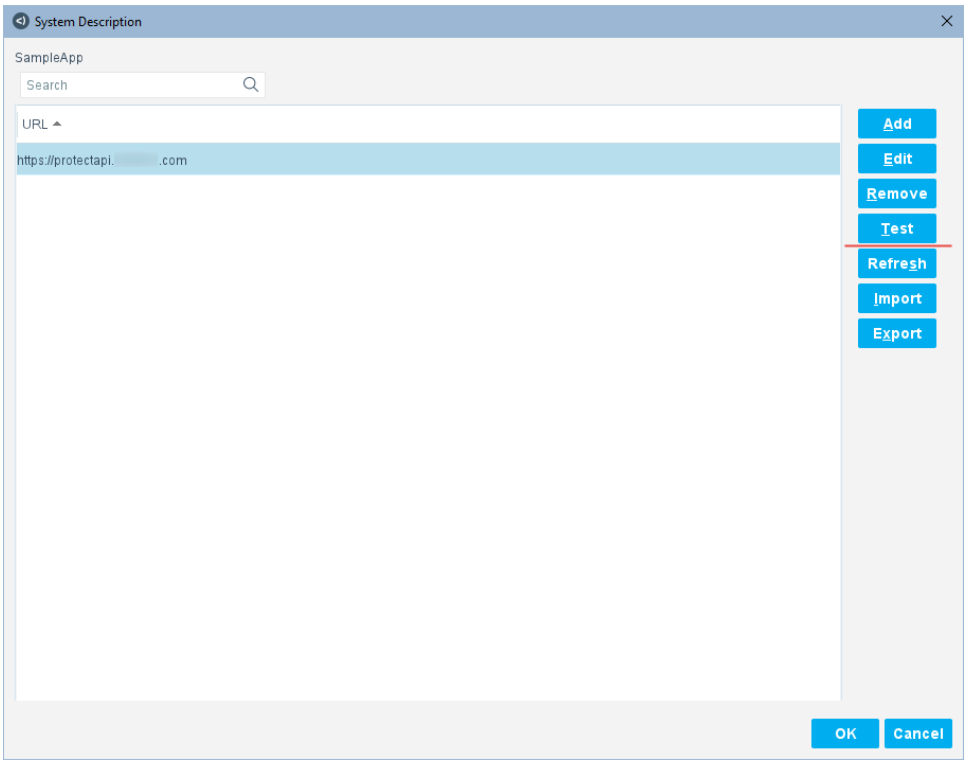
The name, version, and author defined, in the system.conf file, are displayed in the user interface on the Connect pane:



Test button

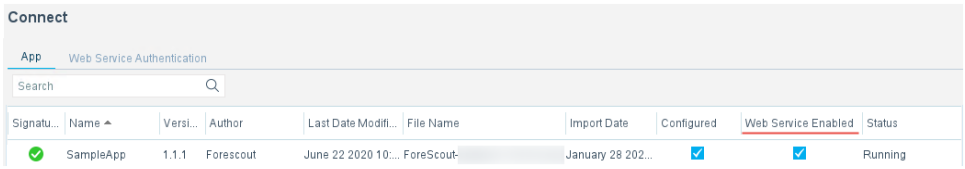
If enabled, the Test button is displayed on the System Description dialog box.

The Test button can be used to test connectivity to a third-party vendor through a Python script:



Web service enabled

If the Connect web service is enabled, there is a check box in the Web Service Enabled column in the Connect pane:



Parameter details for fields

The parameters in the system.conf file are as follows:

Parameter	Description
"name"	(Required) The name of the app. Each app must have a unique name. Names are used in various places in configuration files and Python scripts. Select a specific name rather than a generic name. See App Name Details for rules and examples.

Parameter	Description
"version"	(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
"author"	(Required) The author of the app. The author can contain letters (uppercase and lowercase), numbers, spaces, and special characters such as underscore (_), hyphen (-), percentage (%), period (.), and dollar sign ($).
"testEnable"	(Optional) The device test button. When set to true, a Test button is displayed in the System Description dialog box.
"web service"	(Optional) The Connect web service. When set to true, the web service is enabled. The default is false. See Use the Connect Web Service for details.
"focal only syslog"	(Optional) The syslog focal appliance setting. When set to true, indicates that only the focal appliance will receive the syslog message. When set to false or when not specified in the system.conf file, indicates that all appliances will receive the syslog message (the default). In addition, for the syslog feature, you will need to specify syslog-related settings in the user interface. See Syslog Source Panel Details.

App name details

App names are required and must meet the following rules:

- Can contain letters (uppercase and lowercase), numbers, spaces, and underscores (_)
- Must start and end with an alphanumeric character (uppercase or lowercase)
- Can contain single spaces and/or single underscores separating the alphanumeric characters
- Cannot contain leading or trailing spaces or underscores
- Cannot contain repeated spaces or underscores

If an app name does not follow the rules, the app cannot be imported. Error messages are displayed with details showing the reason the app name is invalid.

Examples of valid app names are:

- Cylance
- Sample App
- Sample_App

- SampleApp
- Sample App123
- Sample_App 123
- cylance123App
- VMware AirWatch

Examples of invalid app names are:

- _ Cylance (contains a leading underscore)
- sample app_ (contains a trailing underscore)
- Sample__App (contains repeated underscores)
- Sample (contains a leading space)
- Sample 123 (contains repeated spaces)
- Sample& (contains an unsupported special character, &)
- Sample#123 (contains an unsupported special character, #)
- sample-app (contains an unsupported special character, -)

A version of the app name is used in the prefix of field names. See Field Name Details for rules and examples.

Edit fields

Specify a name, version, and author by entering the text within the quotation marks, for example:

```
"name":"SampleApp",
"version":"1.0.0",
"author":"Thomas Smith",
```

Edit "testEnabled"

(Optional) To display the Test button on the System Description dialog box, set "testEnable" to true as follows:

```
"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.

Edit "web service"

(Optional) To enable the Connect web service, set "web service" to true as follows:

```
"web service":true,
```

Edit "focal only syslog"

(Optional) To specify if only the focal appliance will receive the syslog message, set "focal only syslog" to true as follows:

```
"focal only syslog": true,
```

To specify if all appliances will receive the syslog message, either set "focal only syslog" to false or remove "focal only syslog" from the system.conf file. The default is for all appliances to receive and process the syslog message.

Note:
Syslog is configured for each app.

Appliance Getting the Syslog

Syslog can be sent to all appliances or only the focal appliance. If there are multiple system descriptions with "focal only syslog" set to true (since it is configured for each app), only the focal appliance will get the syslog message.

In the following example, there are two system descriptions (1 and 2) for an app and there are three appliances (A, B, and C):

- Appliance A is the focal appliance (which is also the default appliance). It has system description 1 with syslog source I defined.
- Appliance B is the assigned appliance (which means that it is the focal appliance for this system description). It has system description 2 with syslog source II defined.
- Appliance C is not used for any system descriptions (it is not a default appliance or an assigned appliance).

If "focal only syslog" is set to true, appliance A gets the syslog message from syslog source I and appliance B gets the syslog message from syslog source II. Appliance C does not get any syslog messages.

If "focal only syslog" is not set or set to false, appliances A, B and C get the syslog message from syslog source I and syslog source II.

The following table provides details.

False or True	Configuration	Description
focal only syslog is false (or not specified)…		When a syslog source is specified, the syslog message is sent to all appliances.
	When there is one system description for the app…	Only one syslog source can be specified. All appliances get the syslog message from that syslog source.

False or True	Configuration	Description
	When there are multiple system descriptions for the app…	All appliances get all syslog messages from all the syslog sources specified. You add syslog sources by adding system descriptions.
	When the focal appliance changes in the system description from A to B…	All appliances will get all syslog messages. (Syslog is not affected by the focal appliance change on the existing system description).
	When the system description is removed…	The syslog source is unregistered to all appliances. All appliances no longer get the syslog message from the removed syslog source(s).
focal only syslog is true…		When a syslog source is specified, the syslog message is sent to the focal appliance only.
	When there is one system description for the app…	Only one syslog source can be specified. The focal appliance gets the syslog message from the syslog source.
	When there are multiple system descriptions for the app…	Each focal appliance handles syslog messages from the syslog source specified. Since the syslog source is tied to the focal appliance, this limits the amount of syslog sources that can be specified to the number of focal appliances defined. The appliance specified in the system description gets the syslog message from syslog source specified in that system description.
	When the focal appliance changes in the system description from A to B…	The syslog source defined in the system description is unregistered to appliance A and the syslog source is registered to appliance B. (Appliance B gets the syslog message and A does not.)
	When the system description is removed…	If the removed system description had focal appliance A and syslog source I, the syslog source I is unregistered to focal appliance A. (Appliance A no longer gets the syslog message from syslog source I.)

Define user interface panels and fields in system.conf

This topic describes how to define the rest of the system.conf file.

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 five panels. The first panel is a custom panel with eight fields. The first four fields on that panel are defined using several parameters, while the fifth, sixth, seventh, and eighth fields on that panel are

predefined. The next three panels (the second, third, and fourth) are predefined. The fifth panel is a custom panel with two predefined fields.

```
    {
       "focal appliance":true,
       "title":"Assign CounterACT Devices",
       "description":"<html>Select the connecting CounterACT device that will communicate with the targeted
       SampleApp instance, including requests by other CounterACT devices. Specific CounterACT devices assigned
       here cannot be assigned to another server elsewhere.<br><br>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.<html>"
    },
    {
       "proxy server":true,
       "title":"Proxy Server",
       "description":"<html>Select a Proxy server to manage all communication between CounterACT and
       SampleApp.</html>"
    },
    {
       "syslog source":true,
       "title":"Syslog Source",
       "description":"<html>Define a Syslog source that sends syslog message to Forescout.</html>"
    },
    {
       "title":"SampleApp Options",
       "description":"SampleApp Options",
       "fields": [
         {
            "host discovery": true,
            "display":"Discovery Frequency",
            "max":300000,
            "add to column":"true",
            "show column":"false",
            "value":3600
         },
         {
            "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
         }
       ]
    }
  ]
}
```

Second panel, which is predefined

Third panel, which is predefined

Fourth panel, which is predefined

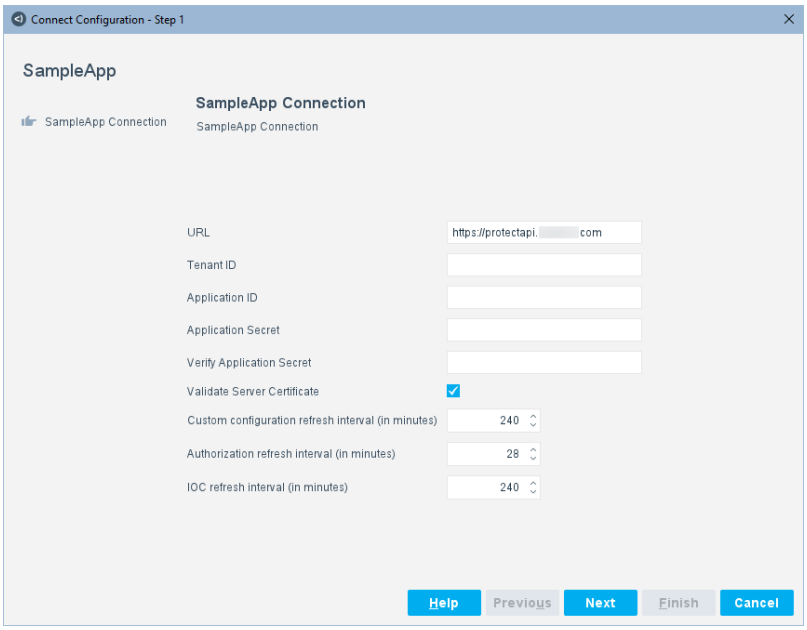
Fifth panel, which is a custom panel with two fields

First field on fifth panel, which is predefined

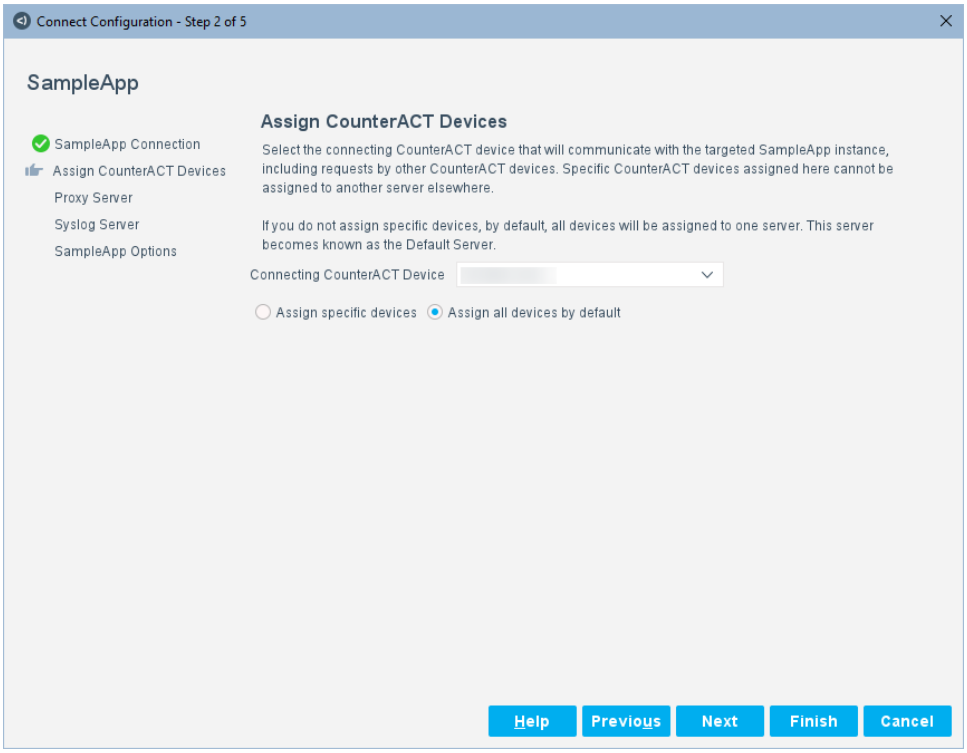
Second field on fifth panel, which is predefined

Panels and fields in user interface

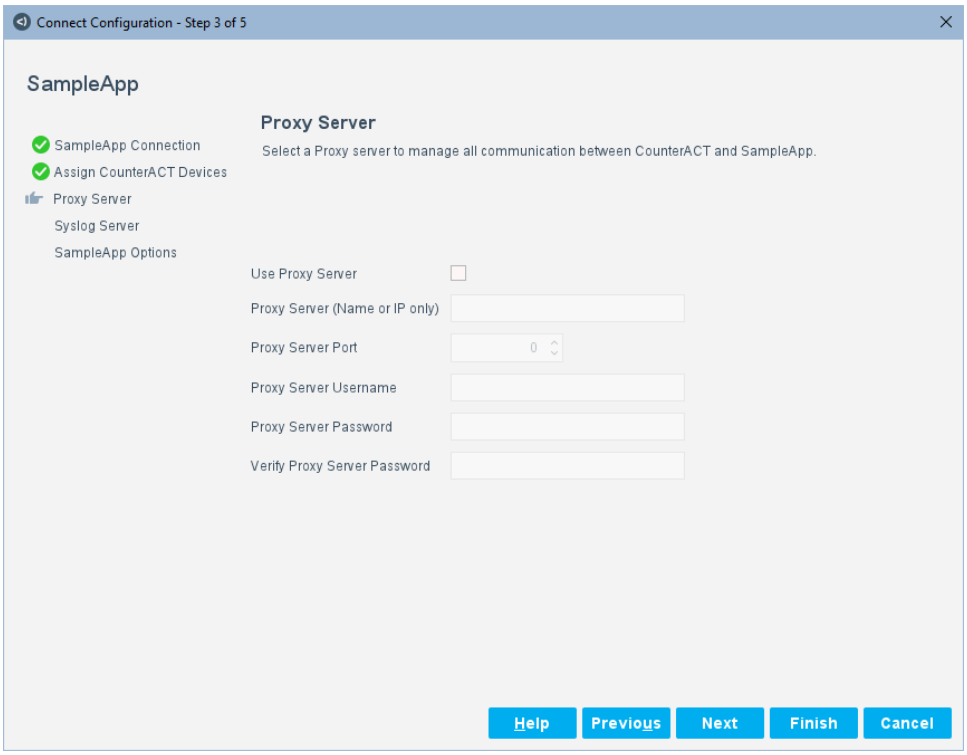
When you select Add in the System Description dialog box, the first configuration panel defined in the system.conf file is displayed. For example, the following panel has several fields as well as a check box:



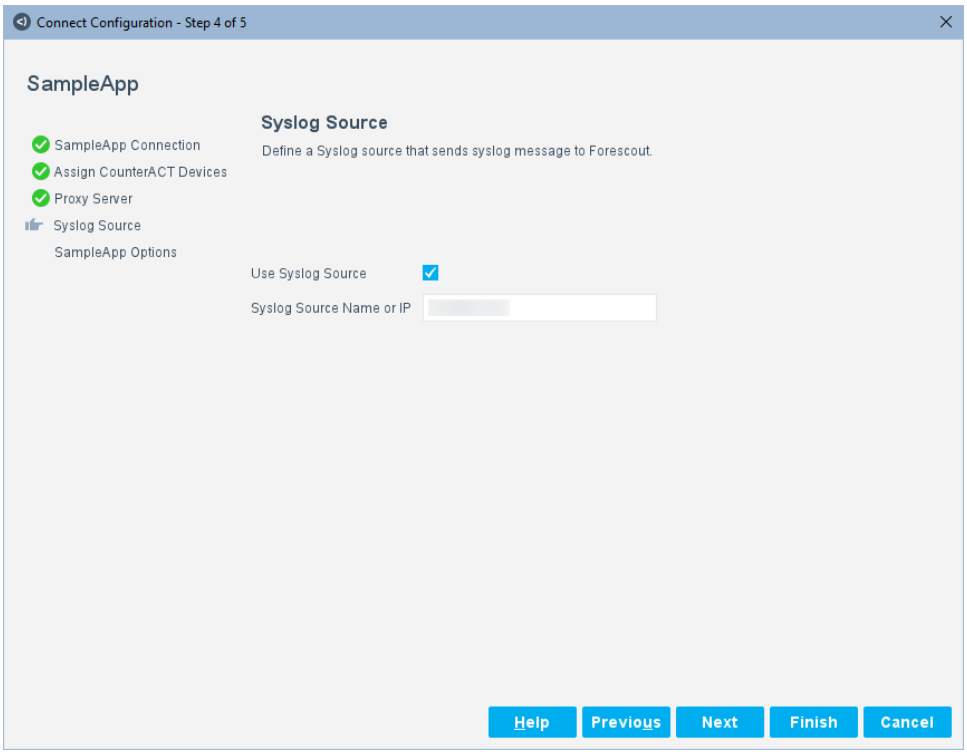
By selecting Next, the second configuration panel defined in the system.conf file is displayed. For example, the predefined Assign CounterACT Devices panel:



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



When you select Next, the fourth configuration panel defined in the system.conf file is displayed. For example, the predefined Syslog Source panel:



When you select Next, the fifth configuration panel defined in the system.conf file is displayed. For example, the custom SampleApp Options panel. See "rate limiter" Field and "host discovery" Field for this panel.

After the last panel, select Finish. The system description is configured.

Parameter details for panels and fields

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

Panel	Field	Description
"panels"		(Required) The definition of the configuration panels in the user interface. In this sample configuration, there are five panels: • "title":"SampleApp Connection"—a custom panel • "focal appliance":true—a predefined panel • "proxy server":true—a predefined panel • "syslog source":true—a predefined panel • "title":" SampleApp Options"—a custom panel The value is a JSON array. Each element of the JSON array is a JSON object.
	"title"	(Required) The title of a custom panel.
	"description"	The description of a custom panel.
	"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. It must start with *connect_<appname>_* and both <appname> and "field ID" must follow a specific format. See Field Name Details for rules and examples. 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.

Panel	Field	Description
		• "identifier"—When set to true, the value of the field must be unique, for example, 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 pre-populate the Cylance URL, use: "value":"https://protectapi.cylance.com" The predefined fields are: • "certification validation"—When set to true, the predefined Validate Server Certificate check box 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 refresh interval field is displayed. See "authorization" Field. • "ioc_poll" —When set to true, the predefined IOC refresh interval field for IOC data is displayed. See "ioc_poll" Field.
"focal appliance"		(Required) The predefined focal appliance panel. When 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. When 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.
"syslog source"		(Optional) The predefined syslog source panel. When set to true, a syslog source panel is displayed as a configuration panel. See Syslog Source Panel Details.

Panel	Field	Description
	"title"	(Required) The title of the panel, which is the predefined syslog source panel.
	"description"	The description of the syslog source panel. HTML formatting can be used in the description, for example, to create multiple paragraphs.

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's recommended to put the most important fields in the first panel, such as IP address, URL, or user name/password.

You can define many parameters for each field, as the 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.

In the following sample configuration, a SampleApp Connection panel is defined with four fields. The definition of each field uses different parameters. See Parameter Details for Panels and Fields.

```
{
   "name":"SampleApp",
   "version":"1.1.1",
   "author":"Concert Masters",
   "testEnable":true,
   "web service":true,
   "focal only syslog": true,
   "panels":[
      {
         "title":"SampleApp Connection",
         "description":"SampleApp Connection",
         "fields":[
            {
               "display":"URL",
               "field ID":"connect_sampleapp_url",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"true",
               "identifier":"true",
               "tooltip":"URL"
            },
            {
               "display":"Tenant ID",
               "field ID":"connect_sampleapp_tenant_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Tenant ID"
            },
            {
               "display":"Application ID",
               "field ID":"connect_sampleapp_application_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Application ID"
            },
            {
               "display":"Application Secret",
               "field ID":"connect_sampleapp_application_secret",
               "type":"encrypted",
               "mandatory":"true",
               "tooltip":"Application Secret"
            },
```

Panel title and description

First field: a mandatory as well as unique string type of field that will be displayed in the System Description dialog box and available in Add/Remove Columns

Second field: a mandatory string type of field that will not be displayed in the System Description dialog box, but will be available in Add/Remove Columns

Third field: a mandatory string type of field that will not be displayed in the System Description dialog box, but will be available in Add/Remove Columns

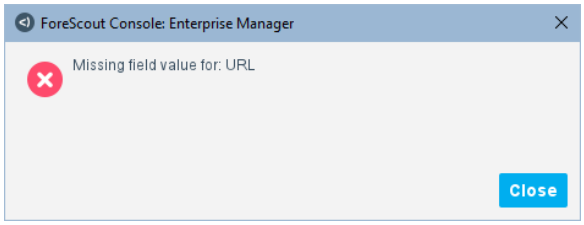
Fourth field: a mandatory field that will not be displayed in the System Description dialog box, nor will it be available in Add/Remove Columns because type is encrypted

"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.

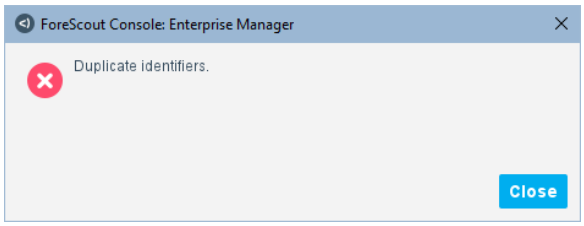


Error message for "identifier" parameter in user interface

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.

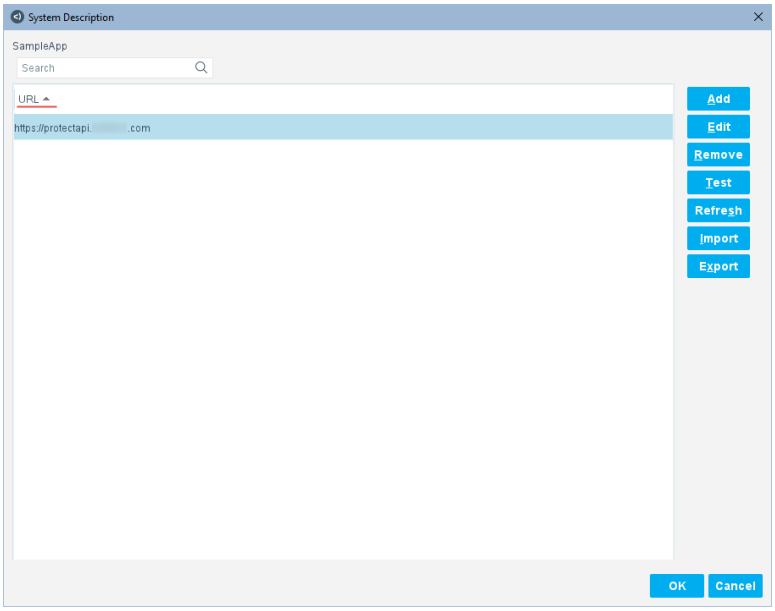
Only one "identifier" is allowed in a system.conf file.

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



"show column" parameter in user interface

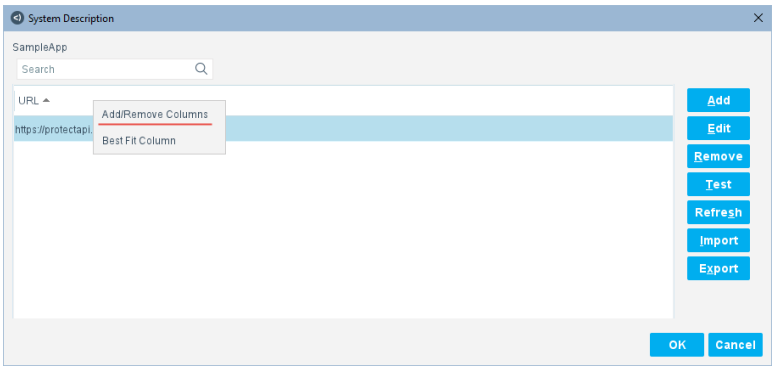
The "show column" parameter results in fields being displayed as columns in the System Description dialog box. In the following example, there is one column.



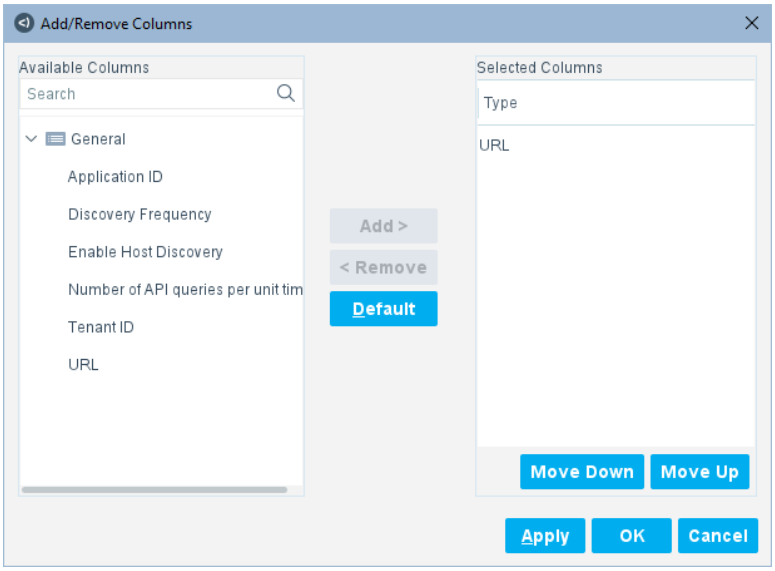
When "show column" is set to true, the field is displayed as a column in the System Description dialog box, if "add to column" is also set to true.

"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.



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.

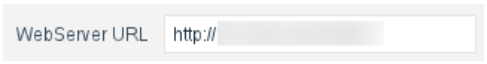
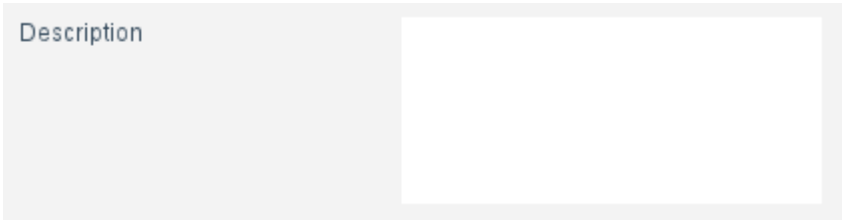
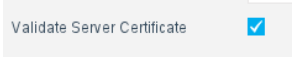
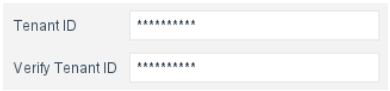
If the field type is “encrypted”, you cannot add it as a column. This prevents the display of passwords in a column.

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

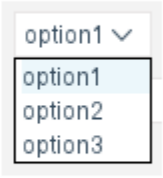
“type” parameter details

The “type” parameters are as follows:

Type	Description
“shortString”	A string field, consisting of one row of editable text. Use this type for a username or a URL.

Type	Description
	
"longString"	A string field, consisting of five rows of editable text. Use this type for a multi-line field, such as a description. 
"ip"	An IP field. Use this type for an IPv4 address.
"integer"	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. 
"boolean"	A Boolean field. Use this type to create a checkbox that takes only a "true" (checked) or "false" (unchecked) value. 
"encrypted"	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.  The two values are checked to see if they match.
"option"	An option field containing a drop-down menu. Use this type for a selectable list. 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	Description

	<pre>"type":"option",
	"options": [
	{
	"display":"option1",
	"value":"OPTION1_FIELD_ID"
	},
	{
	"display":"option2",
	"value":"OPTION2_FIELD_ID"
	},
	{
	"display":"option3",
	"value":"OPTION3_FIELD_ID"
	}
	],</pre> 

"validate client certificate" field

The "validate client certificate" field within the Connect app helps predefined parameters so the plugin can then support any third-party client certificates that are used by all apps.

The following is a sample configuration that can be defined in the system.conf file:

```
{
   "client certification validation":true
},
{
   "display": "Client Certificate Key",
   "field ID": "connect_{appname}_client_cert_key",
   "type": "longString",
   "mandatory": "false",
   "tooltip": "Key for the client certificate"
},
{
   "display": "Client Certificate Data",
   "field ID": "connect_{appname}_client_cert_data",
   "type": "longString",
   "mandatory": "false",
   "tooltip": "client certificate"
}
```

The parameters of the "validate client certificate" field are as follows:

Field	Description
"Validate Client	When the checkbox is selected, the plugin will

Field	Description
Certificate"	then support any third-party client certificates that are used by all apps.
"Client Certificate Key"	Provide the related client certification key for configuration.
"Client Certificate Data"	Provide the related client certificate data for configuration.

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 in the *Forescout Platform Administration Guide*.

"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".

```
{
   "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 lets you configure the interval of how often you want to refresh the data.

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.

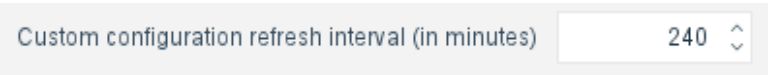
When app instance cache is defined in the system.conf file, the Refresh button on the System Description dialog box is enabled to let you manually trigger a refresh. See Refresh App Features.

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

The parameters of the "app_instance_cache" field are as follows:

Parameter	Description
"app_instance_cache"	(Required) When set to true, indicates an integer field for configuring the app instance cache data refresh interval, in minutes.
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"min"	(Optional) The minimum value of the field. If none is set, "min" defaults to 1.
"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.
"value"	(Optional) The prepopulated default value of the field. If none is set, "value" will be the "min".

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



"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".

```
{
  "authorization":true,
  "display":"Authorization refresh 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 refresh 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.

You also need to specify the script used for authorization. See Authorization Script for Connect. The script must be mapped in the property.conf file. See Map Scripts in property.conf.

When authorization is defined in the system.conf file, the Refresh button on the System Description dialog box is enabled to let you manually trigger a refresh. See Refresh App Features.

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

The parameters of the "authorization" field are as follows:

Parameter	Description
"authorization"	(Required) When set to true, indicates an integer field for the authorization refresh interval (in minutes).
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"min"	(Optional) The minimum value of the field. If none is set, "min" defaults to 1.
"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.
"value"	(Optional) The prepopulated default value of the field. If none is set, "value" will be the "min".

The resulting field, Authorization refresh interval (in minutes), is displayed in the user interface.

"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".

```
{
    "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.

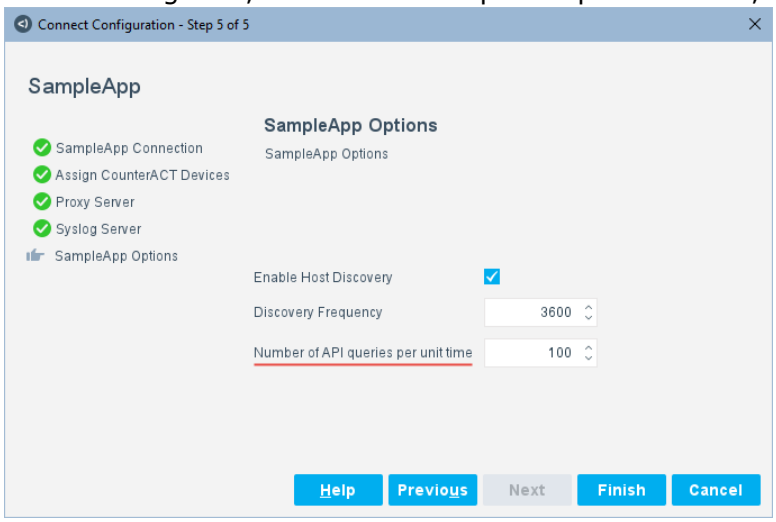
Only one "rate limiter" field is allowed in a system.conf file.

The parameters of the "rate limiter" field are as follows:

Parameter	Description
"rate limiter"	(Required) When set to true, indicates an integer field for the rate limiter.
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"unit"	(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.
"min"	(Optional) The minimum value of the field. If none is set, "min" defaults to 1.
"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.

Parameter	Description
"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" field

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

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

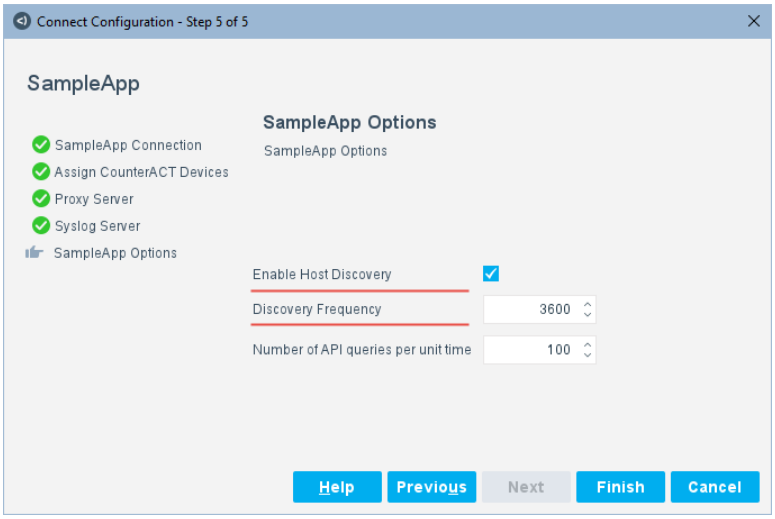
Only one "host discovery" field is allowed in a system.conf file.

The parameters of the "host discovery" field are as follows:

Parameter	Description
"host discovery"	(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.

Parameter	Description
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"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" defaults to 1. The unit is minutes.

The resulting fields are displayed in the user interface. To enable the Discovery Frequency field, select the Enable Host Discovery checkbox.



You also need to specify the script used for host discovery. The script must be mapped in the property.conf file. See Map Scripts in property.conf.

When discovery is defined in the system.conf file and the Enable Host Discovery checkbox is selected, the Refresh button on the System Description dialog box is enabled to let you manually trigger a refresh. See Refresh App Features.

"ioc_poll" field

The following is a sample configuration of an Indicators of Compromise (IOC) poll field that can be defined in the system.conf file. It uses a predefined parameter named "ioc_poll".

```
{
   "ioc_poll":true,
   "display":"IOC refresh interval (in minutes)",
   "min":5,
   "max":2400,
   "value":240
}
```

Use the "ioc_poll" field to periodically get updated IOC data from third-party sources through an API call. The IOC polling process is triggered by configuring the IOC refresh interval (a scheduled task) that specifies how often you want to refresh the data.

You also need to specify the script used for retrieving the IOC data from the third-party. The script must be mapped in the property.conf file. See Map Scripts in property.conf.

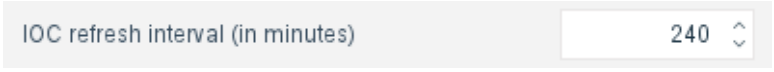
When the IOC poll is defined in the system.conf file, the Refresh button on the System Description dialog box is enabled to let you manually trigger a refresh. See Refresh App Features.

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

The parameters of the "ioc_poll" field are as follows:

Parameter	Description
"ioc_poll"	(Required) When set to true, indicates an integer field for configuring the IOC poll refresh interval, in minutes.
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"min"	(Optional) The minimum value of the field. If none is set, "min" defaults to 1.
"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.
"value"	(Optional) The prepopulated default value of the field. If none is set, "value" will be the "min".

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



"timer" field

The Connect Plugin now allows Connect Apps to define multiple generic timers.

Parameter	Description
"timer"	(Required) When set to true, timer execution occurs and then repeats based on the intervals set by the user.
"field ID"	(Required) To ensure the field ID is defined, see additional required steps in Parameter Details for Scripts.
"display"	(Required) The label of the field, which will be displayed on the left of the field in the user interface.
"min"	(Optional) The minimum value of the field. If none is set, "min" defaults to 1.
"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

Parameter	Description
	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 pre-populated default value of the field. If none is set, "value" will be the "min".

The following is an Example configuration of field that can be defined in the system.conf file:

{ "display":"Example Timer", "field ID":"connect_testapp_timer1", "mandatory":"true", "value":5, "min":1, "max":30, "tooltip":"Timer", **"timer":"true"** }

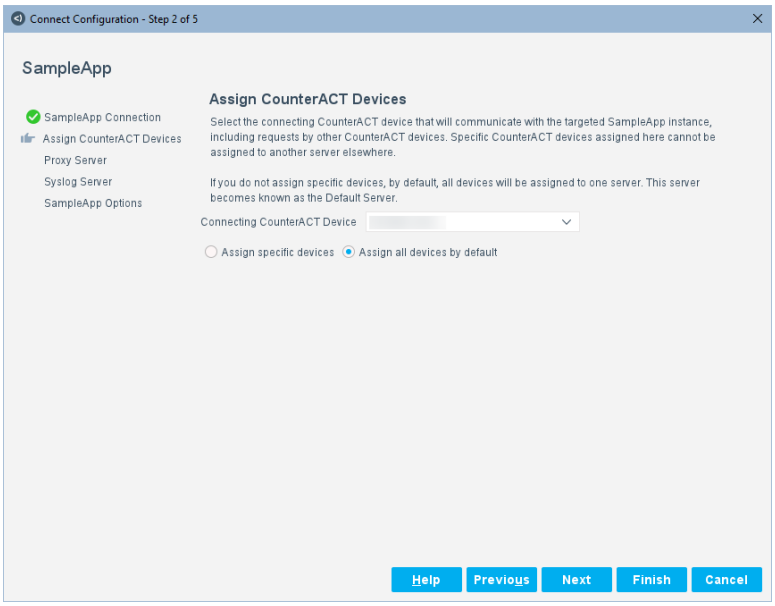
Assign CounterACT devices panel details

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.



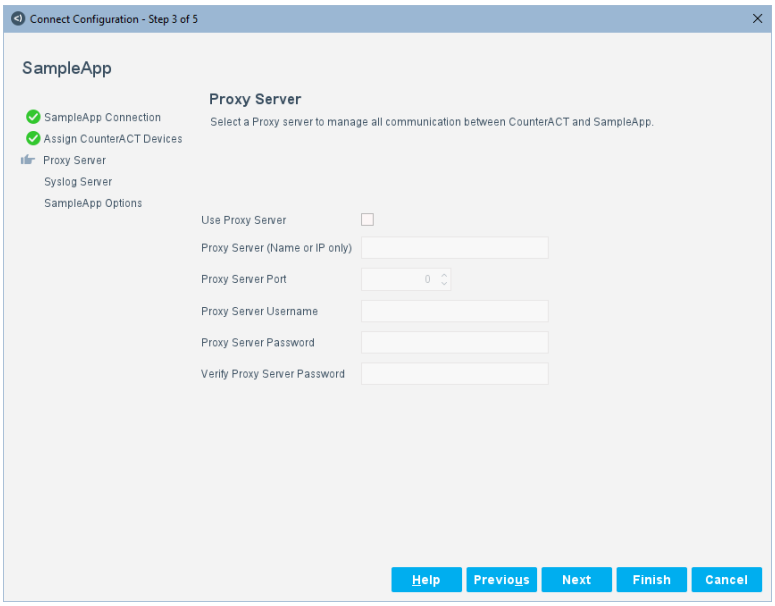
More devices can be added after the first device. Subsequently, the Assign CounterACT Devices panel has more fields. See Add a System Description for details about the predefined fields on the panel.

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 optional. It is predefined and results in the following panel.



If the Use Proxy Server checkbox is selected, some of the remaining fields are required. Both authentication and non-authentication modes are supported, so 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.

Proxy Server Panel Field IDs

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 as follows:

- 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

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

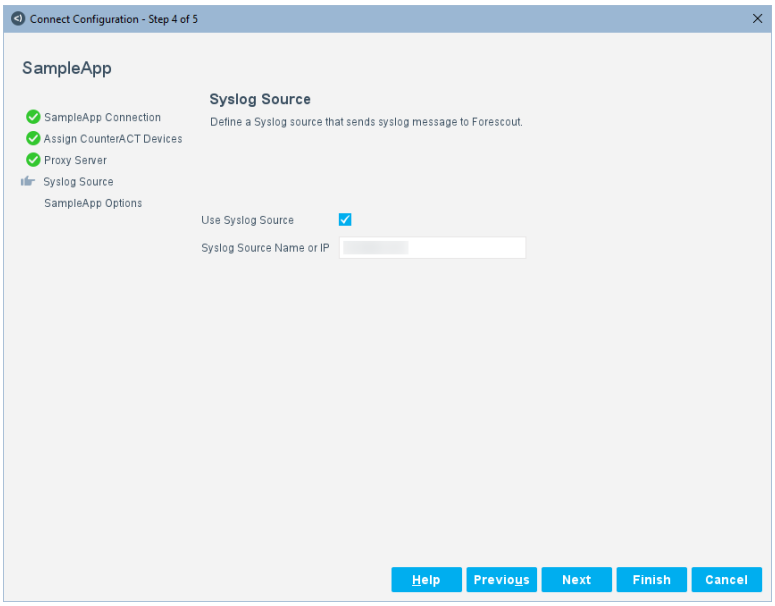
Use Connect Proxy Server Python Module

You can use the Connect Proxy Server module in a script to handle proxy server calls supporting the Proxy Server panel.

For information, see Appendix B: Connect Proxy Server and Sample Connect Script Files

Syslog source panel details

The “syslog source” panel is optional. It is predefined and results in the following panel.



Select the Use Syslog Source checkbox to enable or disable the syslog source as follows:

- If Use Syslog Source is enabled, the Syslog Source Name or IP field is required.
- If Use Syslog Source is disabled, Connect does not get or process the syslog message from the specified syslog source.

Use the Syslog Source Name or IP field to enter the name or IP address of the syslog source on which Connect listens for the syslog message.

Only one syslog source panel is allowed in a system.conf file.

Note:
Different apps can have the same syslog source, but one app cannot have the same syslog source defined more than once.

Summary of system.conf rules

This topic is a summary of the system.conf file rules.

Rule	For More Information
Define at least two panels with one field on each panel	Define Panels and Fields
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

Rule	For More Information
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
Only one "ioc_poll" field is allowed	"ioc_poll" 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 Only one focal appliance panel is allowed Focal appliance panel cannot be the first panel	Assign CounterACT Devices Panel Details
Only one proxy server panel is allowed	Proxy Server Panel Details
Only one syslog source panel is allowed	Syslog Source Panel Details

Leverage Forescout Platform plugins

This topic describes leveraging other plugins.

Some Connect features leverage other Forescout Platform plugins, such as Syslog and IOC Scanner.

Connect Syslog support

Connect can listen to a Syslog source and receive syslog messages by leveraging the Syslog Plugin through Connect configuration.

The syslog feature leverages the Syslog Plugin to pass the syslog message, received from syslog sources to Connect. Connect can expose the syslog message in properties so you can parse the syslog message and use it in a script. One example is to parse the syslog message for possible threats and then send this information to the IOC Scanner Plugin and alert the user.

When syslog is configured in the system.conf and/or property.conf file, and the syslog source name or IP address is specified and enabled in the user interface, Connect is able to receive syslog messages from that syslog source. The syslog message is exposed through a Connect tag letting you further process the syslog message in the script.

Connect does not get the syslog message when:

- The syslog configuration is removed or not set for an app.
- The syslog source is disabled in the system description.
- The app is stopped.
- The Connect Plugin is stopped.

Note:
The Syslog Plugin must be running on the appliances.

For more information on the Syslog Plugin, refer to Forescout Core Extensions Module: Syslog Plugin Configuration Guide.

IOC scanner support

Connect leverages the IOC Scanner Plugin to store and save IOC information into the Forescout Platform from third-party sources. The Connect app parses the threat information received from third-party sources or from syslog and sends the information to the IOC Scanner Plugin.

You need to specify where to get the threat information from the third-party source, whether through an API call or syslog. You can use one (or more) of the following three methods to trigger an IOC update:

- Manually triggered through a policy recheck if the IOC data is polled from a third-party source through an API call
- Periodically triggered by scheduled task through setting up the IOC refresh interval if the IOC data is polled from a third-party through an API call
- Triggered when IOC-related data is retrieved from the syslog message

Note:
The IOC Scanner Plugin must be running on the appliances.

For more information on the IOC Scanner Plugin, refer to Forescout Core Extensions Module: IOC Scanner Plugin Configuration Guide.

Define property.conf file

This topic describes defining the 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 topics:

See Sample property.conf File.

Define name in property.conf

Define the "name" field in the property.conf file.

```
{
  "name": "SampleApp",
```

Parameter details for name

The parameter for "name" is as follows:

Parameter	Description
"name"	(Optional in the property.conf file) The name of the app. The best practice is to use the same name as in the system.conf file. Names are used in various places in configuration files and Python scripts. Select a specific name rather than a generic name. See App Name Details for rules and examples.

Define property groups in property.conf

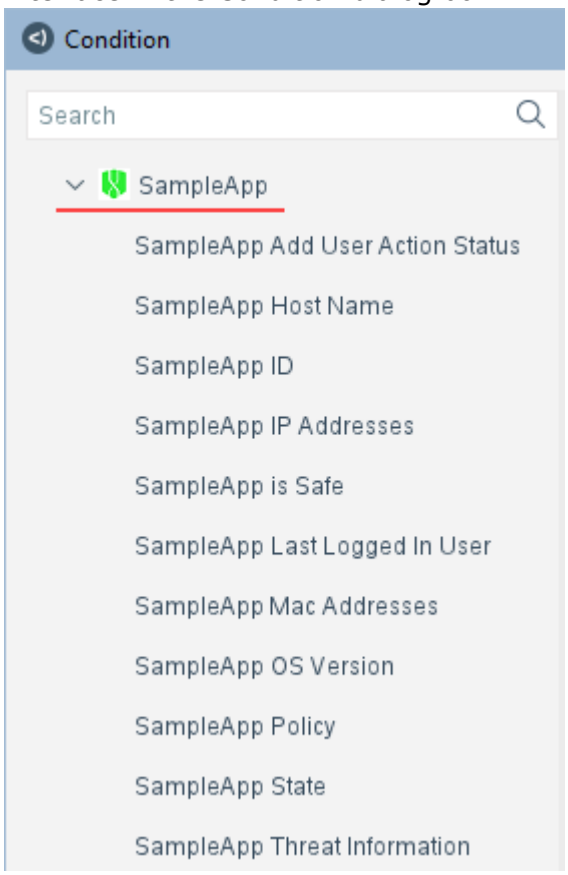
Use "groups" to define property group names. The value of "groups" is a JSON array. Each element of the JSON array is a JSON object.

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

```
"groups": [
  {
    "name":"connect_sampleapp_sampleapp",
    "label":"SampleApp"
  }
],
```

“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:

Parameter	Description
“name”	(Required) The internal, unique name of the group. It must start with *connect_<appname>_*. See Field Name Details for rules and details.
“label”	(Required) The label of the group displayed in the user interface.

Field name details

Field names must follow a format that has three parts consisting of a two-part prefix followed by the name of the field. The first part of the prefix is always *connect_*. The second part of the prefix is based on the app name defined in the system.conf file followed by an underscore. The last part is the name of the field, so the format is: *connect_<appname>_<fieldname>*. A simple example is connect_sampleapp_url.

The two-part prefix starts with *connect_* and is followed by a version of the app name defined in the system.conf file, which must meet the following rules:

- All lowercase letters of the name of the app (see App Name Details)
- No spaces or underscores

The following table lists examples of the name of the app defined in the system.conf file versus the appname as part of the prefix:

Name of the App	appname in Prefix
Cylance	cylance
Sample App	sampleapp
Sample_App	sampleapp
SampleApp	sampleapp
Sample App123	sampleapp123
cylance123App	cylance123app
VMware AirWatch	vmwareairwatch

The third part of the format is the name of the field. Field names are required and must meet the following rules:

- Uppercase and lowercase letters and numbers
- Underscores
- No spaces or special characters

Examples of the field name format, complete with the two-part prefix, are:

- connect_cylance_tenant_ID
- connect_sampleapp_host_name

- connect_sampleapp123_mac_addresses
- connect_cylance123app_add_user_action
- connect_vmwareairwatch_state

If a field name does not follow the rules, the app cannot be imported. Error messages are displayed with details showing the reason the field name is invalid.

Define properties in property.conf

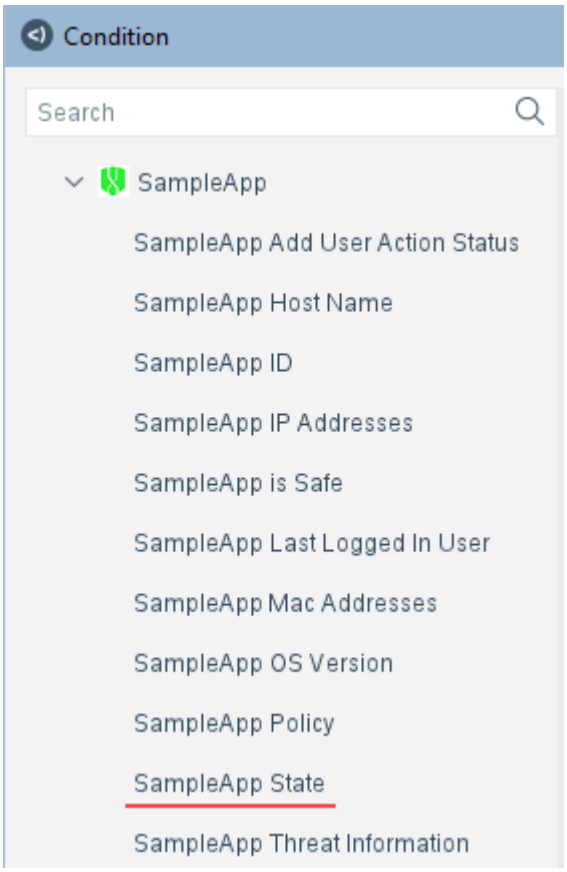
Use “properties” to define the properties that can be used as conditions in policies. The value of “properties” is a JSON array. Each element of the JSON array is a JSON object.

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

```
"properties": [
  {
    "tag": "connect_sampleapp_state",
    "label": "SampleApp State",
    "description": "SampleApp State",
    "type": "string",
    "web_enable": true,
    "options": [
      {
        "name": "Online",
        "label": "Online"
      },
      {
        "name": "Offline",
        "label": "Offline"
      }
    ],
    "group": "connect_sampleapp_sampleapp",
    "resolvable": true,
    "require_host_access": false,
    "inventory": {
      "enable": true,
      "description": "Inventory of SampleApp State"
    },
    "asset_portal": true,
    "track_change": {
      "enable": true,
      "label": "SampleApp State Changed",
      "description": "Track Change property for SampleApp 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:

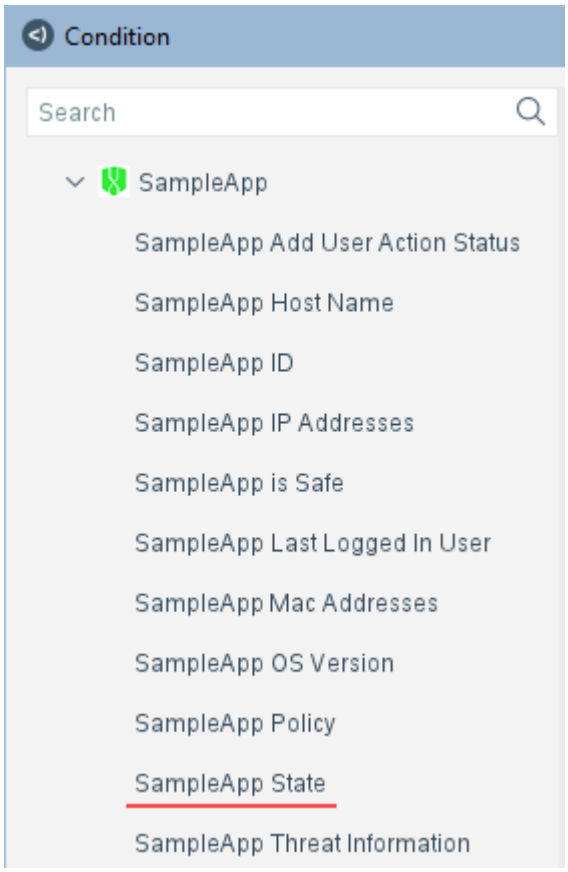
Parameter	Description
"tag"	(Required) The internal, unique name of the property. It must start with *connect_<appname>_*. See Field Name Details for rules and details. If the properties are used in a script to resolve properties, the property tags must be listed in the property.conf file under "scripts". See Parameter Details for Scripts.
"label"	(Required) The label of the property displayed in the user interface. See "label" in User Interface.
"description"	(Required) The description of the property displayed in the user interface. See "description" in User Interface.

Parameter	Description
"type"	(Required) The type of the property. The valid types are string, boolean, integer, date, and composite. See Property "type" Details.
"web_enable"	(Optional) When set to true, this property can be updated through the Connect web service. The default is false. To use the web service API to update a property, set "web_enable" to true. See Use the Connect Web Service for details.
"group"	(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.
"options"	(Optional) The options of a string property type. See "options" in Property "type" Details.
"resolvable"	(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.
"require_host_access"	(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.
"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

Parameter	Description
"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.

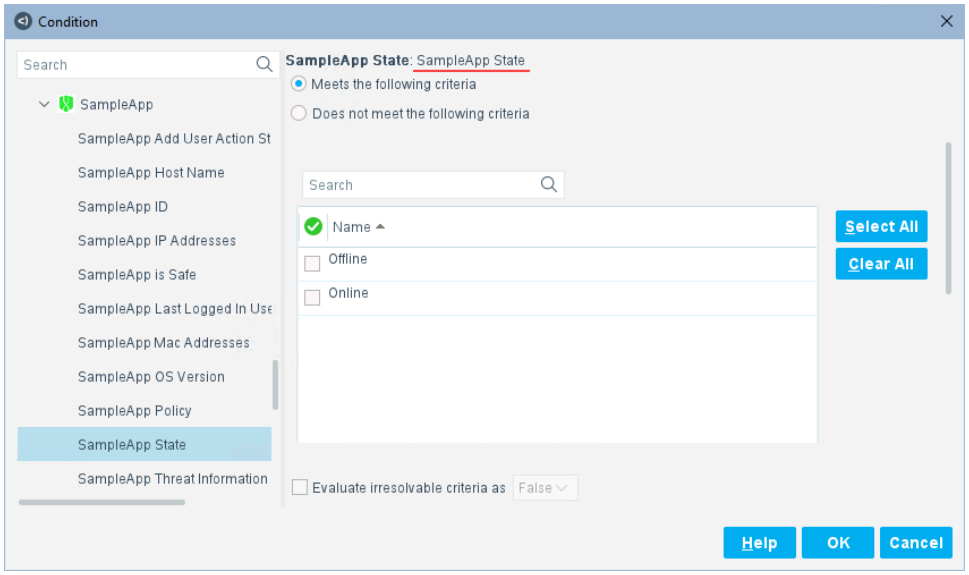
"label" in user interface

The "label" of a property is displayed in the Condition dialog box in the user interface.



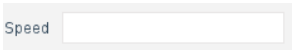
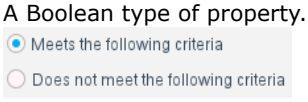
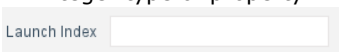
"description" in user interface

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



Property "type" details

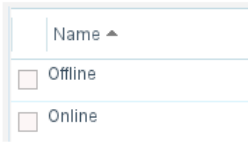
The property "type" parameters are as follows:

Type	Sub-Type	Description
"string"		A string type of property. 
	"options"	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: • "name"—(Required) The value of the option used to resolve the property • "label"—(Required) The label in the user interface See "options" Details.
"boolean"		A Boolean type of property. 
"integer"		An integer type of property. 
"date"		A date type of property.  See "date" Details.
"composite"	"subfields"	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: • "tag"—(Required) The tag of the field, which is a text string, in alphanumeric characters. It must be unique only within the property. • "label"—(Required) The label in the user interface. • "description"—(Required) The description in the user interface. • "type"—(Required) The type of the subfield. The valid types are string, integer, boolean, and date. • "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. See "composite" Details.

"options" details

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

```
"type": "string",
"options": [
  {
    "name": "Online",
    "label": "Online"
  },
  {
    "name": "Offline",
    "label": "Offline"
  }
],
```



“date” details

When populating the "date" property type, use epoch time.

The following two examples use the datetime library in Python scripts:

Example 1:

```
prop_val = "2021-02-17T09:40:30Z"
prop_val = int(datetime.strptime(prop_val, '%Y-%m-
%dT%H:%M:%SZ').strftime('%s'))
```

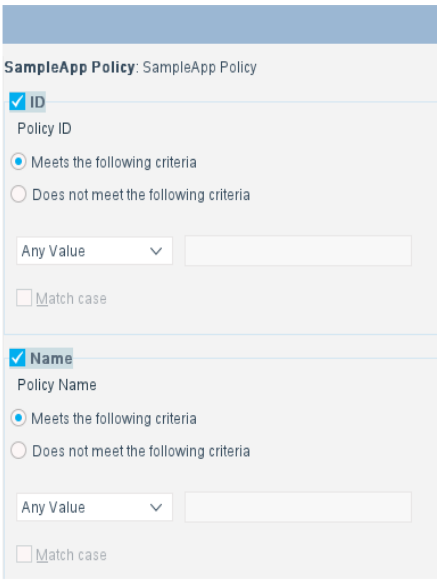
Example 2:

```
time_str = "02/17/2021 10:00:00"
epoch = int(datetime.strptime(time_str, '%m/%d/%Y %H:%M:%S').timestamp())
```

“composite” details

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

```
{
  "tag": "connect_sampleapp_policy",
  "label": "SampleApp Policy",
  "description": "SampleApp Policy",
  "type": "composite",
  "group": "connect_sampleapp_sampleapp",
  "inventory": {
    "enable": true,
    "description": "Inventory of SampleApp 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
    }
  ]
},
```



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

```
{
  "tag": "connect_sampleapp_policy",
  "label": "SampleApp Policy",
  "description": "SampleApp Policy",
  "type": "composite",
  "group": "connect_sampleapp_sampleapp",
  "inventory": {
    "enable": true,
    "description": "Inventory of SampleApp 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
    }
  ]
},
```

Enable Inventory on property

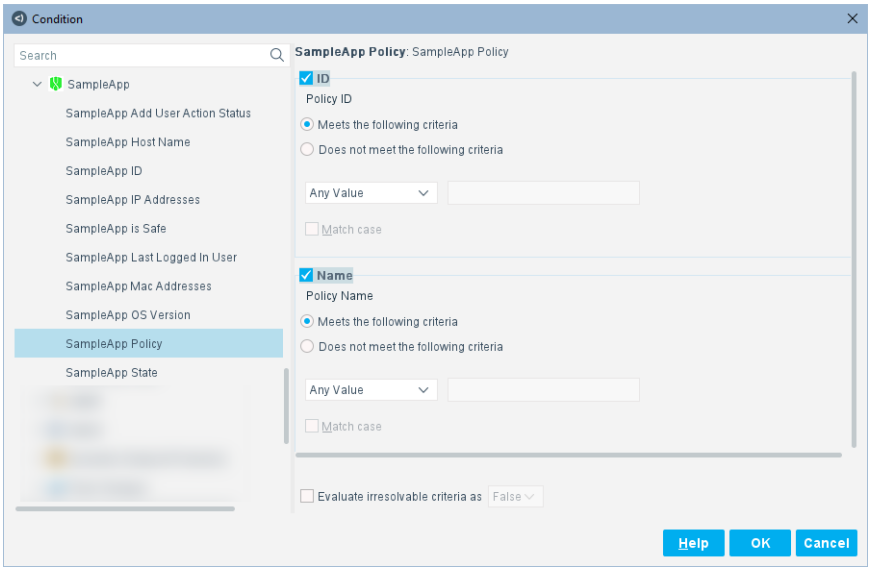
Enable Inventory on subfield

Enable Inventory on subfield

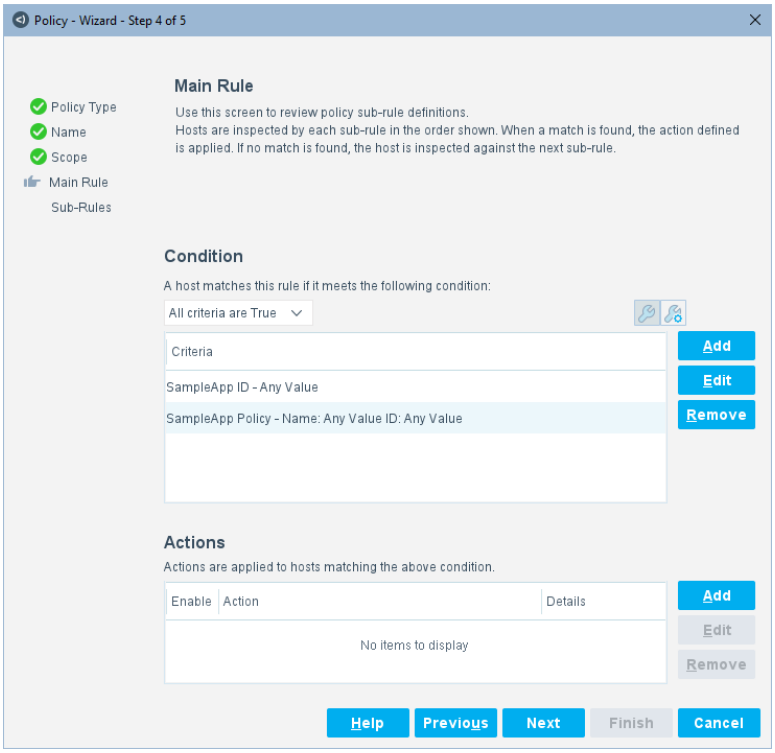
Composite subfields cannot be added as dependencies. However, you can access the subfields if you add the parent composite property as a dependency.

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

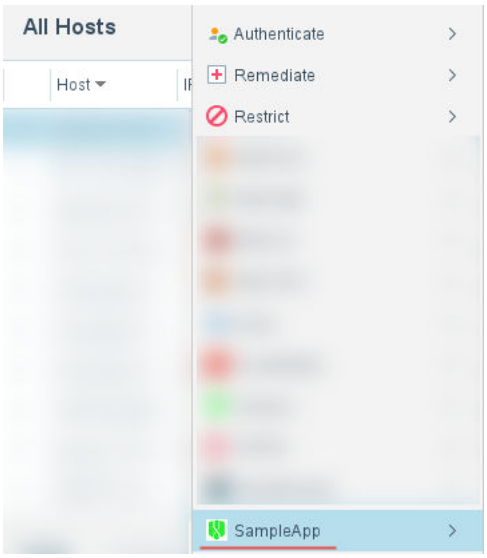
Use "action_groups" to define action group names. The value of "action_groups" is a JSON array. Each element of the JSON array is a JSON object.

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

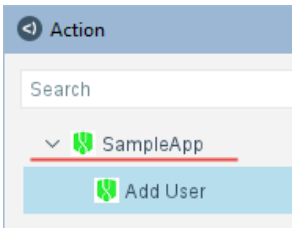
```
"action_groups": [
  {
    "name":"connect_sampleapp_sampleapp",
    "label":"SampleApp"
  }
```

"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 Action in the Action dialog box.



Parameter Details for Action Groups

The parameters for "action_groups" are as follows:

Parameter	Description
“name”	(Required) The internal, unique name of the action group. It must start with *connect_<appname>_*. See Field Name Details for rules and details.
“label”	(Required) The label of the action group displayed in the user interface.

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 canceled either manually or if the policy no longer applies.

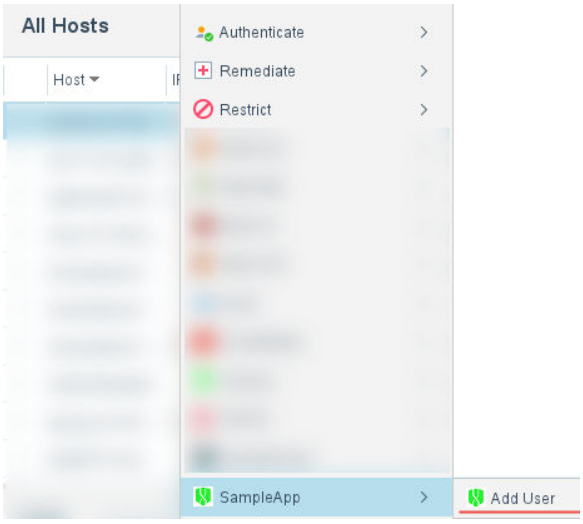
Use “actions” to define actions. The value of “actions” is a JSON array. Each element of the JSON array is a JSON object.

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

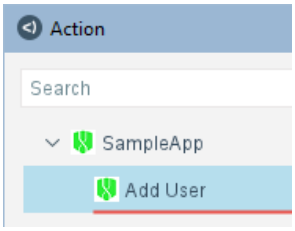
```
"actions": [
  {
    "name": "connect_sampleapp_add_user",
    "label": "Add User",
    "group": "connect_sampleapp_sampleapp",
    "description": "Add New User",
    "ip_required": false,
    "threshold_percentage": 1,
    "params": [
      {
        "name": "sampleapp_email",
        "label": "Email address",
        "description": "SampleApp email address",
        "type": "string"
      },
      {
        "name": "sampleapp_first_name",
        "label": "First name",
        "description": "SampleApp first name",
        "type": "string"
      },
      {
        "name": "sampleapp_last_name",
        "label": "Last name",
        "description": "SampleApp last name",
        "type": "string"
      }
    ],
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ],
    "undo": {
      "label": "Cancel SampleApp 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 Action in the *Action* dialog box.



Parameter details for actions

The parameters for “actions” are as follows:

Parameter	Description
“name”	(Required) The internal, unique name of the action. It must start with *connect_<appname>_*. See Field Name Details for rules and details. If the actions are used in a script to execute actions, the action names must be listed in the property.conf file under “scripts”. See Parameter Details for Scripts.
“label”	(Required) The label of the action displayed in the user interface.
“description”	(Required) The description of the action.
“group”	(Required) The group to which the action belongs. This must match a name defined for “action_groups” or the name of an existing action group in the Forescout Platform.
“ip_required”	(Optional) When set to true, the action cannot be taken when the IP address is missing on the host. The default is false.
“threshold_percentage”	(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.
“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.

Parameter	Description
	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.d

"params" parameter details

The "params" parameters are as follows:

Parameter	Description
"name"	(Required) The unique name of the parameter, in alphanumeric 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, boolean, and list. The list type is used to select the value from a drop-down list without typing the whole string. It has a sub-type of "options" to define the list options. The value of "options" is a JSON array. Each element of the JSON array is a JSON object with the following fields: • "name"—(Required) The value of the option used to resolve the property • "label"—(Required) The label in the user interface The following is a sample of the list type:

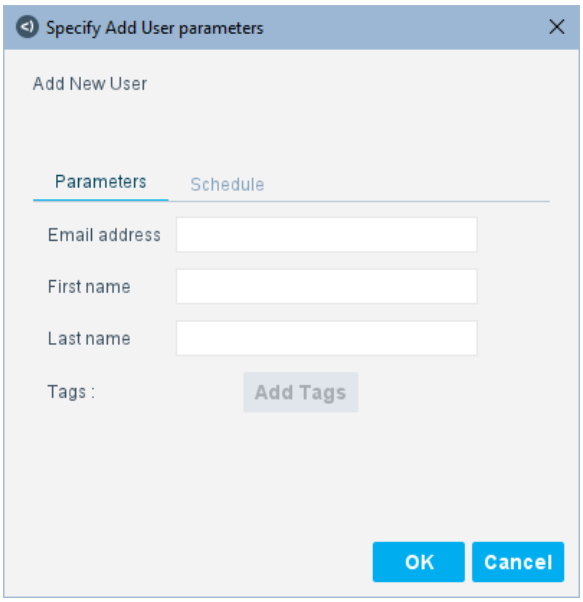
Parameter	Description

	<pre>"type": "list",
	"options": [
	  {
	  "name": "activescan",
	  "label": "Active Scan"
	  },
	  {
	  "name": "fullscan",
	  "label": "Full Scan"
	  }
	]</pre>

"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.

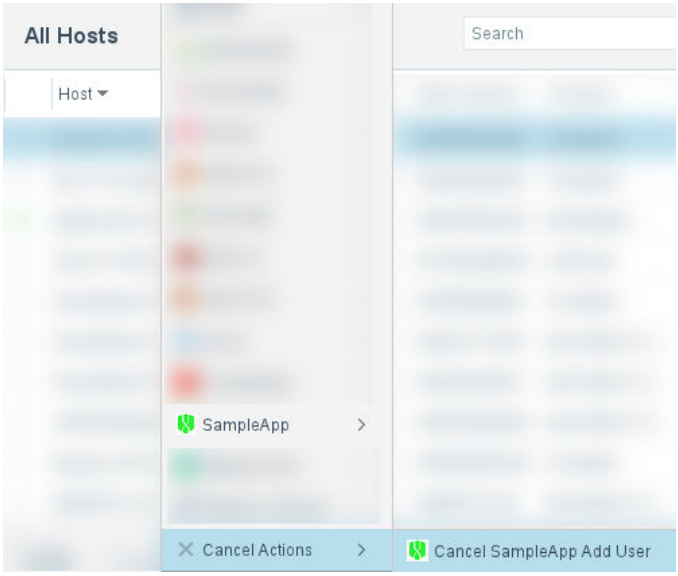
"params" in user interface

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



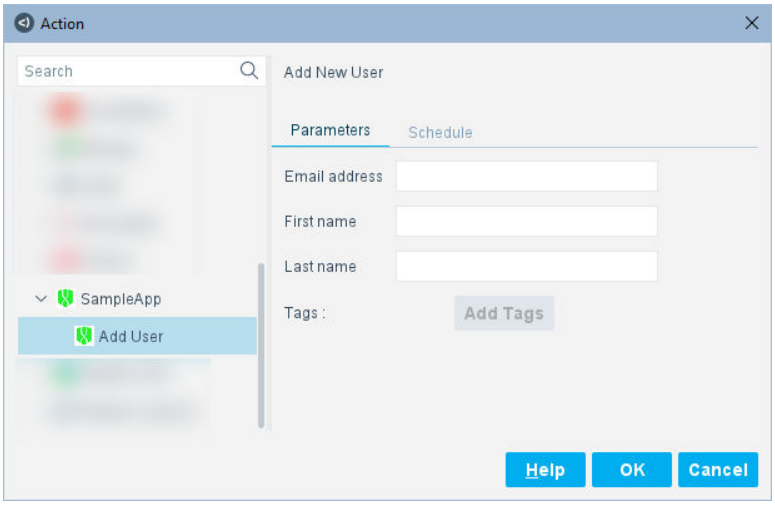
“undo” in user interface

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.

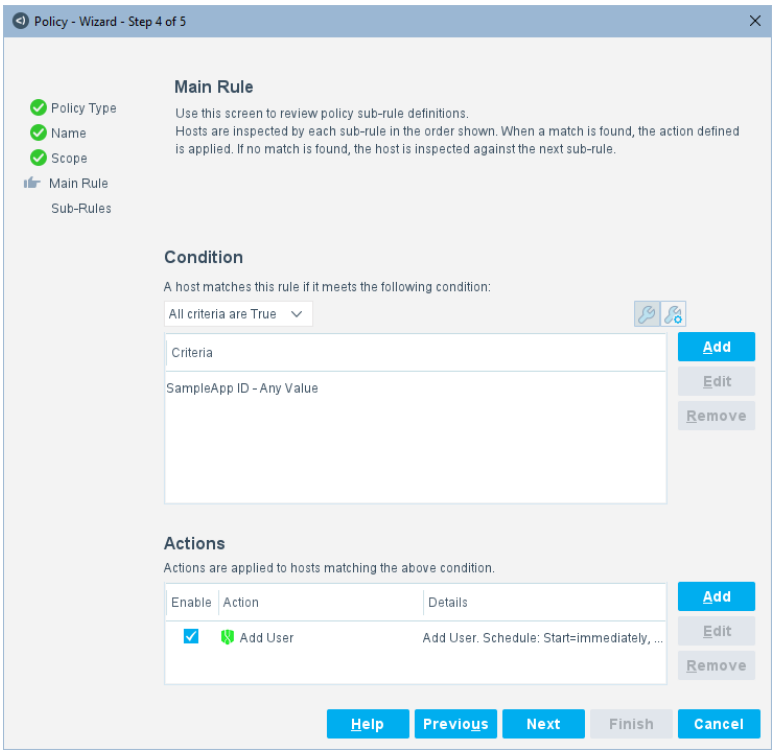


Actions in policy templates

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:

Parameter	Description
"name"	(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.
"timer"	When set to true, timer execution occurs and then repeats based on the intervals set by the user by selecting the "Refresh" button.

Parameter	Description
"field ID"	(Required for setting timers) The field ID property must be set to the same field ID defined in the system.conf file. Example: • *system.conf* file ID: "connect_testapp_timer1" • *property.conf* file parameters:{ "display":"Example Timer", **"field ID":"connect_testapp_timer1"**, "mandatory":"true", "value":5, "min":1, "max":30, "tooltip":"Timer", "timer":"true" }
"properties"	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.
"actions"	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.
"is_cancel"	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.
"test"	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.
"discovery"	When set to true, the script discovers hosts from a third-party source. See Polling Script for Connect.
"authorization"	When set to true, the script gets authorizations. See Authorization Script for Connect.
"app_instance_cache"	When set to true, the script gets app instance cache data. See Use App Instance Cache in Connect Scripts.
"library_file"	When set to true, you can put your own Python files to serve as library files within an app. See Python Library Files.
"ioc_poll"	When set to true, the script gets IOC data from a third-party source. See Use IOC Poll in Connect Scripts.
"syslog_message"	When set to true, the received syslog message is available to be used in the script. See Use Syslog Message in Connect Scripts.

Parameter	Description
"batch_size"	The maximum batch size (1-50) for running action and property resolves in one script execution. See Batching of Connect App Action and Property Resolves.d

"scripts" details

The following sample "scripts" in the property.conf file shows the mapping of ten different scripts.

```
"scripts": [
  {
    "name": "sampleapp_resolve.py",
    "properties": [
      "connect_sampleapp_state",
      "connect_sampleapp_last_logged_in_user",
      "connect_sampleapp_mac_addresses",
      "connect_sampleapp_is_safe",
      "connect_sampleapp_id"
    ]
  },
  {
    "name": "sampleapp_ioc_resolve.py",
    "properties": [
      "connect_sampleapp_host_name",
      "connect_sampleapp_os_version"
    ]
  },
  {
    "name": "sampleapp_add_user.py",
    "actions": [
      "connect_sampleapp_add_user"
    ]
  },
  {
    "name": "sampleapp_delete_user.py",
    "is_cancel": true,
    "actions": [
      "connect_sampleapp_add_user"
    ]
  },
  {
    "name": "sampleapp_test.py",
    "test": true
  },
  {
    "name": "sampleapp_poll.py",
    "discovery": true
  },
  {
    "name":"sampleapp_authorization.py",
    "authorization":true
  },
  {
    "name":"sampleapp_parse_syslog_message.py",
    "syslog_message":true
  },
  {
    "name":"sampleapp_users.py",
    "app_instance_cache": true
  },
  {
    "name":"sampleapp_ioc_poll.py",
    "ioc_poll": true
  }
],
```

first script — resolve properties script, which needs property tags

second script — IOC resolve properties script, which needs property tags

third script — add user action script, which needs action names

fourth script — cancel add user action script

fifth script — test script

sixth script — discovery script

seventh script — authorization script

eighth script — syslog parsing script

ninth script — app instance cache script

tenth script — IOC poll script

Define policy templates

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 topic.

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

Use "policy_template" to define policy templates.

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

Parameter	Description
"policy_template_group"	(Required) The policy template group in the app. See Define Policy Template Group in property.conf.
"policies"	(Required) The policy templates in the app. See Define Policies in property.conf.

Define policy template group in property.conf

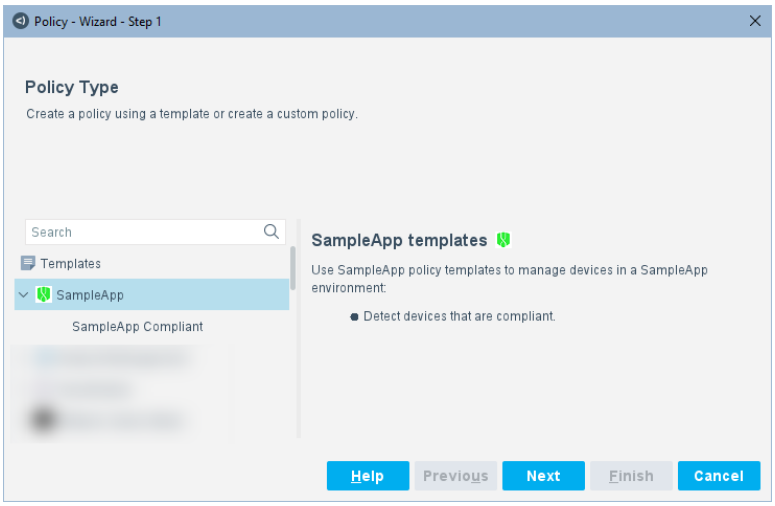
Use "policy_template_group" to define the policy template group. The value of "policy_template_group" is a JSON object.

Only one policy template group can be defined.

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

Parameter	Description
"name"	(Required) The internal, unique name of the policy template group. It must start with *connect_<appname>_*. See Field Name Details for rules and details.
"label"	(Required) The label of the policy template group displayed in the user interface.
"display"	(Required) The heading of the policy template group displayed in the user interface.
"description"	(Optional) The description of the policy template group displayed in the user interface.
"full_description"	(Optional) The full description of the policy template group displayed in the user interface. HTML formatting can be used in the full description, for example, to create a bulleted list.
"title_image"	(Optional) The icon image for the policy template group. See Define Icons in Connect.

Define policies in property.conf

You can customize a third-party vendor integration by defining policy templates.

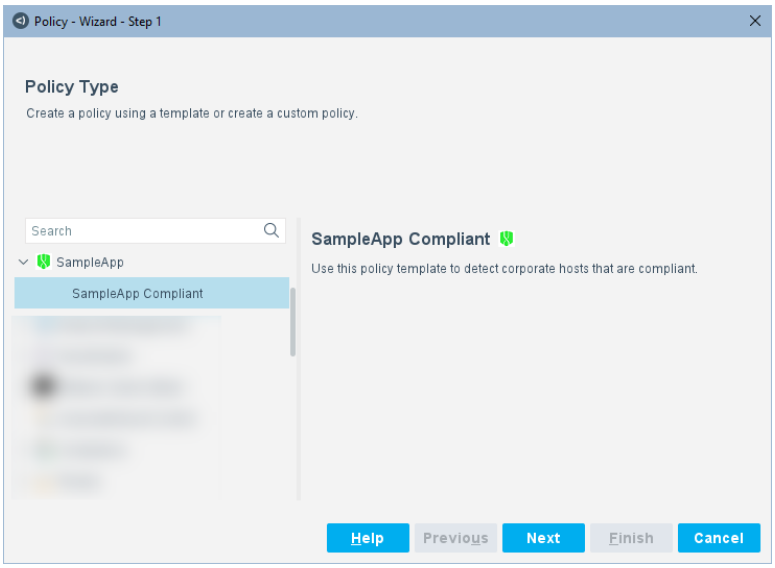
Use "policies" to define the policy templates. The value of "policies" is a JSON object.

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

```
"policies": [
  {
    "name": "connect_sampleapp_compliant",
    "label": "SampleApp Compliant",
    "display": "SampleApp Compliant",
    "help": "SampleApp Compliant Policy",
    "description": "Creates SampleApp compliant policies",
    "file_name": "SampleAppCompliance.xml",
    "full_description": "<html>Use this policy template to detect
    corporate hosts that are compliant.</html>",
    "title_image": "connect_sampleapp.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:

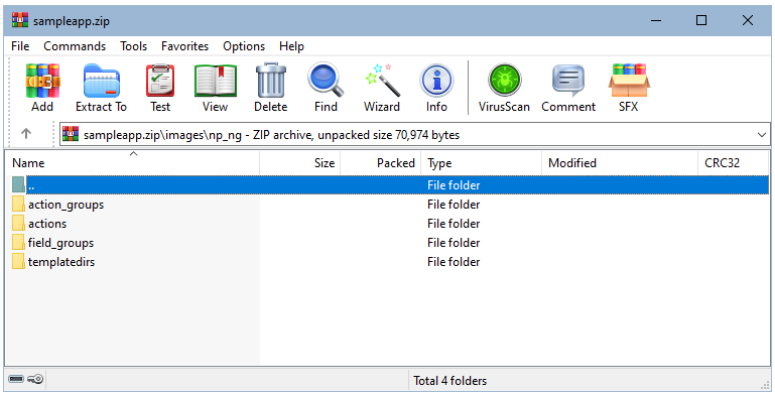
Parameter	Description
“name”	(Required) The internal, unique name of the policy template. It must start with *connect_<appname>_*. See Field Name Details for rules and details.
“label”	(Required) The label of the policy template displayed in the user interface.
“display”	(Required) The heading of the policy template displayed in the user interface.
“description”	(Optional) The description of the policy template displayed in the user interface.

Parameter	Description
"help"	(Optional) The help information of the policy template, which is displayed when you hover the mouse over the Help button in the user interface.
"file_name"	(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.
"full_description"	(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.
"title_image"	(Optional) The icon image for the policy template. See Define Icons in Connect. d

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.



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

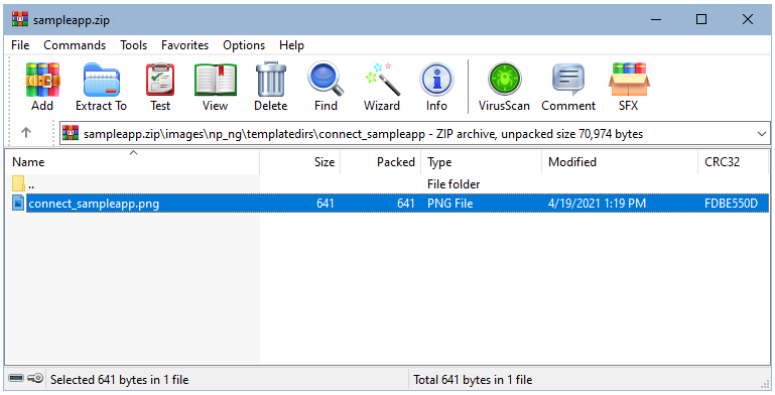
```
"title_image": connect_sampleapp.png
```

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:

`<appname>/images/np_ng/templatedirs/connect_<appname>/connect_<appname>.png`

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

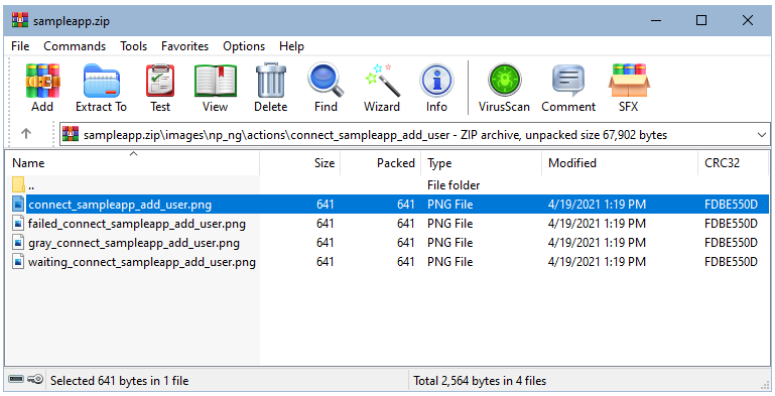
For other policy template icons, put them in the folder: `<appname>/images/np_ng/templatedirs/connect_<appname>`



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

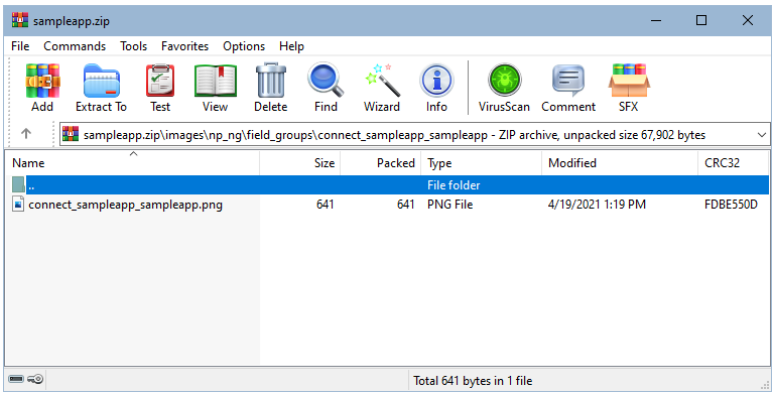
`<appname>/images/np_ng/actions/connect_sampleapp_add_user`

For action icons, the icon names have a specific format. For example, if the action icon is named connect_sampleapp_add_user.png, the icon for the corresponding failed action must have the same name but be prefixed with *failed_*. Similarly, use the prefixes *gray_* and *waiting_* for those action states.

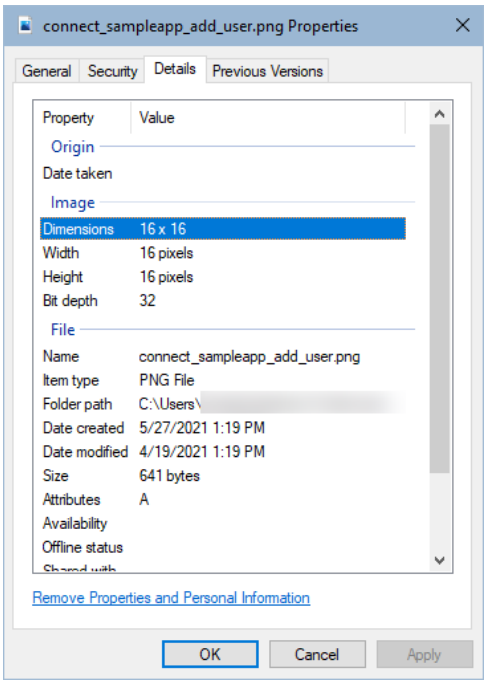


For action group icons, each group defined in the property.conf file must have a folder in the action_groups folder. The folder name and image name should be the same as the name defined in the property.conf file for “action_groups”.

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

This topic describes how to create a policy template file for Connect.

A policy template file in .xml format should be created before completing the policy template definition in the property.conf file.

If you have three different policy templates, you'll also 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)
    d. Select Next
    e. In the Main Rule pane, select properties and actions, and then select Next
    f. (Optional) In the Sub-Rules pane, select properties and actions, and then select Next
    g. Select Finish
5. When the custom policy template is created, it'll display in the Policy Manager. Select it in xml format and select Export
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

This topic describes writing 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.
- Authorization. See Authorization Script for Connect and Token-Based Authorization.

Examples of using predefined fields in scripts are:

- App instance cache. See Use App Instance Cache in Connect Scripts.
- IOC poll. See Use IOC Poll in Connect Scripts.
- Parse syslog messages. See Use Syslog Message in Connect Scripts.
- Validate Certification. See Use Certificate Validation in Connect Scripts.
- IOC response. See Format Data to "ioc" Response in Resolve, IOC Poll, and Syslog Parsing Scripts.

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

About Python scripting for Connect

The following sections provide information about Python scripting for Connect:

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

About Python libraries

Connect uses Python version 3.6 or 3.8 depending on the version installed in the appliance. It has built-in libraries that you can import. Due to security implications, do not import other libraries.

The following links contain the standard built-in libraries that can be imported into scripts:

https://docs.python.org/3.6/library/ or https://docs.python.org/3.8/library/

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

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

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

- _dummy_thread
- _thread
- cmd
- ctypes
- dnspython
- dns
- dummy_threading
- eval
- exec
- fcntl
- glob
- imp
- importlib
- io
- multiprocessing
- open
- os
- pathlib
- pdb

- platform
- shutil
- subprocess
- sys
- threading
- trace
- traceback
- tracemalloc

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

- eval
- exec
- open
- print

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.

Python 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 located in the following path:

`/usr/local/forescout/plugin/connect_module/python_logs`

Python script not found

The Python scripts included in "scripts" in 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 dialog box and then select Apply in the Connect pane to save the system description configuration.

For example, the SampleApp 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.

Connect response objects

See the following response objects for test scripts:

- Mandatory Fields
- Examples

Mandatory fields

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

Examples

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

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": "sampleapp_add_user.py",
  "actions": [
    "connect_sampleapp_add_user"
  ]
},
{
  "name": "sampleapp_delete_user.py",
  "is_cancel": true,
  "actions": [
    "connect_sampleapp_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.

Connect response objects

See the following response objects for action scripts:

- Mandatory Fields
- Examples for Cookie
- Examples for Properties

Mandatory fields

The mandatory fields for action scripts are as follows.

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

Examples for cookie

Examples for the cookie in response objects for action scripts are as follows.

```
response = {"succeeded" : True}
response = {"succeeded" : True, "cookie" : 13452829256}
response = {"succeeded" : False, "troubleshooting" : "Action failed.
Response code: 402."}
```

Examples for properties

Examples for properties in response objects for action scripts are as follows.

```
response = {"succeeded" : True,
"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"
                }
        ]
}
```

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:

```
sampleapp_to_ct_props_map = {
    "state": "connect_sampleapp_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.

Update IP Address using MAC address as key

You can update the IP address of an endpoint in resolve scripts using the MAC address as a dependency by adding both "mac" and "ip" key/value pairs in the response. For example:

```
response = { "mac": "8058f8c7887b", "ip": "10.0.0.1",
    "properties": {
          "property1": "property_value1",
          "property2": "property_value2"
    }
}
```

To provide access to the MAC address with params["mac"] / params.get("mac") in the resolve script, the properties being resolved should have MAC address as a dependency in the property.conf file. See "dependencies" in the following sample:

```
{
  "tag": "connect_intune_nac_azure_device_id",
  "label": "Intune NAC Azure Device ID",
  "description": "Intune NAC Unique identifier for the device",
  "type": "string",
  "group": "connect_intune_nac_attributes",
  "require_host_access": false,
  "resolvable": true,
```

```
  "asset_portal": false,
  "dependencies": [
    {
      "name": "mac",
      "redo_new": true,
      "redo_change": true
    }
  ]
}
```

Turn host online or offline

You can turn a host online or offline by adding “online” as the key to the properties in the response. The value is boolean. For example:

```
response = {
    "properties": {
          "online": False,
          "property1": "property_value1",
          "property2": "property_value2"
    }
}
```

When “online” is True, it turns the host online if it was previously offline. When “online” is False, it turns the host offline if it was previously online. Without adding “online”, there is no change in state from the original state.

Get IOC data from third-party to send to IOC scanner

In addition to properties, you also get updated IOC data from third-party sources and send it to the IOC Scanner Plugin during resolve. Use your own codes to get IOC data to pass to the “ioc” response dictionary.

To pass IOC data, the response dictionary must contain a dictionary or a list of dictionaries called "ioc", which contains IOC information (single IOC or multiple IOCs).

For details about the "ioc" response format, see Format Data to "ioc" Response in Resolve, IOC Poll, and Syslog Parsing Scripts.

Response objects

See the following response objects for property resolve scripts:

- Mandatory Fields
- Examples

Mandatory fields

The mandatory fields for property resolve scripts are as follows.

```
"properties": a map/dictionary that contains host properties; the key will
be the property name and the value will be the property value.
"ioc": a map/dictionary or a list of maps/dictionaries that contains IOC
data. Add this response object if you want to update IOC data during
resolve. This is an optional field.
"error": Error message if script fails for a known reason
```

Examples

Examples of response objects for property resolve scripts are as follows.

```
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"
                }
        ]
}
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"
                }
        ]
"ioc":{
        "name": "name_value",
        "file_name": "file_name_value",
        "hash": "hash_value",
        "severity": "severity_value"
}
}
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"
                }
         ]
"ioc": [
{
                "name": "name_value_1",
                "file_name": "file_name_value_1",
                "hash": "hash_value_1",
                "severity": "severity_value_1"
```

```
},
{
                "name": "name_value_2",
                "file_name": "file_name_value_2",
                "hash": "hash_value_2",
                "severity": "severity_value_2"
}
]
}
```

Batching of Connect app action and property resolves

This section describes updates in Action/Property Resolve batching within Connect Apps.

App developers can now batch multiple action and property resolves in one script execution within Connect Apps, allowing for more efficient API calls.

When creating a script, users can add a 'batch_size' key to the script definition that ranges in value from 1 to 50. This then indicates the maximum batch that is sent to the Python script. See Parameter Details for Scripts for further information on script keys & parameters.

Action Resolves

Example code for action resolves:

- 'batch_size' set to 50

```
logging.debug(str(endpoints))
logging.debug(str(params))
response = {}
for i in endpoints:
    cid = i["correlation_id"]
    itemresponse = {}
    itemresponse["succeeded"] = True
    response[cid] = itemresponse
```

Property Resolves

Example code for resolving a property:

- 'batch_size' set to 50

```
logging.debug(str(endpoints))
logging.debug(str(params))
response = {}
for i in endpoints:
```

```
cid = i["correlation_id"]
properties = {}
properties[<Property Name>] = "<Property Value>"
itemresponse = {}
itemresponse["properties"] = properties
response[cid] = itemresponse
```

In the examples, the endpoints array contains all endpoint-specific properties:

- Dependencies
- Action parameters
- Correlation ID

The action script will loop through all endpoints and provide one response for each correlation ID. The value for each correlation_id and default_response key should match the format as a non-batched action or resolve scripts.

Note:
Developers can also fill in the response ["default_response"], which overrides the default response for correlation IDs skipped by the script.

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

See the following response objects for authorization scripts:

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.

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.

You can turn a host online or offline by adding "online" as the key to the properties in the response. The value is boolean. For example:

```
properties["online"] = False
response["properties"] = properties
```

When "online" is True, the host is turned online. When "online" is False, the host is turned offline. The default is True.

See Sample Polling Script for Connect.

Response objects

See the following response objects for polling/discovery scripts:

- Mandatory Fields
- Mandatory Sub-Fields for Endpoints
- Examples

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
```

Response object examples

Examples of response objects in polling/discovery scripts are as follows.

```
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"}
       }
    ]
}
```

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 IOC poll in Connect scripts

You can write a script to get IOC data from the third-party source. To parse and process the data into the correct IOC format, see Format Data to "ioc" Response in Resolve, IOC Poll, and Syslog Parsing Scripts.

The IOC field for the response object is "ioc". 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 IOC Poll Script for Connect.

Use Syslog message in Connect scripts

You can write a script to parse the message to extract the information needed. As in this example, the script parses syslog messages to extract threat-related information and format them to an IOC scanner data format that can be sent to the IOC Scanner Plugin. The syslog message content is in string format.

The syslog message can be retrieved by calling params.get("connect_syslog_message"). If available, the syslog message is stored as a string. To parse and process the data into the correct IOC format, see Format Data to "ioc" Response in Resolve, IOC Poll, and Syslog Parsing Scripts.

The IOC field for the response object is "ioc". Put the extracted value from the syslog message that is to be sent to IOC in this field. If there is any error, put "error" in the response object. For an example, see Sample Parse Syslog Message Script for Connect.

In addition to sending IOC data to the IOC Scanner, you can resolve IOC properties using extracted values from the syslog message for an endpoint. Define IOC properties in the “properties” field of the property.conf file. To resolve IOC properties, the response must include the “properties” field, which is a dictionary of IOC property key-value pairs. The “mac” or “ip” of the endpoint is required to pass into the response object so that Connect knows the endpoint for which it should resolve the properties.

And, in addition to resolving IOC properties, you can resolve other properties created in the app, similar to a property resolve script. To create a new host when the host in the script response does not exist, add the "online" field to be true in the properties field. Otherwise, only existing hosts will be updated.

For example:

```
response["ip"]:"x.x.x.x"
response["properties"]:
    {
       "online":True,
       "connect_sampleapp_threat_info":
        [
            {"severity":"low",
             "date":1624382227,
             "file_name":"cscript.exe",
             "name":"attackframework",
             "hash":"2e8d4a6fedf90e265d2370a543b4fd2a",
             "hash_type":"md5"}
        ]
    }
response["ioc"]:
```

```
{
    "severity": "low",
    "file_exists":{
        "file_path":"c:\\windows\\system32\\",
        "file_name":"cscript.exe"
    },
    "file_name":"cscript.exe",
    "name":"attackframework",
    "platform":"none",
    "hash":"2e8d4a6fedf90e265d2370a543b4fd2a",
    "hash_type":"md5"
}
```

For an example, see Sample Parse Syslog Message Script for Connect.

Use Persistent Data feature in Connect apps

Users can now utilize the Persistent Data feature, which allows data to be kept across Connect events allowing for data caching, model transformations, and metrics among other uses.

To utlizie this feature, users must import "PersistUtil" class from the "persistent_util" connect library and initializes it with connect_app_name attribute.

Note:

connect_app_name holds the actual app name of this user app, which will be a user's database name.

For example:

```
from persistent_util import PersistUtil
logging.debug(f"Got app_name {connect_app_name}")
mydb_util = PersistUtil(connect_app_name)
```

For call methods of 'mydb_util', ect.:

```
mydb_util.save("cached", "subscriber_api_key",
api_details["subscriber_api_key"])   #(table, key_str, value_str) tuple
save_result = mydb_util.save("param-tests", "params",
str(params))                    #save_result indicates if this call is
successful
saved_key = mydb_util.get("cached",
"subscriber_api_key")                                          #retrieve
value of a key from a table "cached"
logging.debug(f"api key saved: {saved_key}")
saved_params = mydb_util.get("param-tests", "params")
```

```
logging.debug(f"saved parameters {saved_params}")

mydb_util.delete("param-tests", "params")  #deletes key "params" from
"param-tests" table
mydb_util.cleanup_table("param-
tests")
      # delete the whole table "param-tests"
```

There are some restrictions within this util that user's should be aware of:

1. Only string value can store in the database
2. Max of 10 tables per app
3. Each table has a max size of 100M (i.e. file size)

Additional methods available from the PersistUtil class:

Methods	Description
cleanup_table(table)	Deletes an individual table's 'table' of this app
save(table, key, value)	Saves a single key and value into the 'table' • Both key and value can only be string • 'value' can be strung json object, nested k-v included • Returns as True/ False
get(table, key)	Retrieves the value of a certain 'key' from the 'table'
get_all_keys(table)	Returns all the keys of a 'table' in the form of a list

Methods	Description
get_all_data(table)	Returns all the data in this 'table'
delete(table, key)	Deletes a specific 'key'
__repr__	For debugging purposes only, to print the persistent database name
cleanup_app_data()	Cleans up all the persistent data that pertains to the app

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:

```
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:

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

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

Note:
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).

Format data to "ioc" response in resolve, IOC poll, and Syslog parsing scripts

In the script, put the IOC data that will be sent to the IOC Scanner Plugin in the "ioc" field in the response object. If there is any error, put "error" in the response object.

The value of response["ioc"] could be a dictionary for a single IOC or list of dictionaries for multiple IOCs.

For each IOC dictionary, there are the following:

- Mandatory fields: “name”, “file_name”, “hash”, and “severity”
- Optional fields: “date”, “hash_type”, “platform”, and IOC type: “cnc”, “dns”, “file_exists”, “process”, “mutex”, “registry”, or “service”

  The IOC types show different types of IOCs detected during the lifecycle of a threat.

For example:

```
response["ioc"] = {
    "name": "ioc_test",
    "date": 1621571367000,
    "hash_type": "ShA256",
    "hash":
"4747d091f230bf409af6a42c32cca30d845541d2c3200bb275f0fa179bd5e999",
    "severity": "medium",
    "file_name": "test_file_1.exe",
    "file_size": 44,
    "platform": "Windows",
    "cnc": "10.1.1.10",
    "dns": ["query1", "query2", "query3"],
    "process": {
        "process_name": ["test_process_1", "test_process_2"],
        "process_hash": ["process_hash_1","process_hash_2"],
        "process_hash_type": ["sha1", "none"]
    },
    "file_exists": {
        "file_name": ["test_file_1.exe"],
        "file_path": ["C:\\test1"]
    },
    "mutex": "mutex_value",
    "service": ["service_value1", "service_value2"]
    "registry": {
        "registry_element": "registry_element_value",
        "registry_data": "registry_data_value"
    }
}
```

Field descriptions for IOC dictionary

The fields and descriptions for the IOC dictionary are as follows:

Field	Description
"name"	(Required) The threat name. The name is used for threat identification purposes only.
"date"	The date and time when a threat with this file hash was first reported. If it is not specified in the "ioc" response, the current date is used.
"hash"	(Required) The hash value of the threat file, which is used to identify threats, in hexadecimal format, using lowercase characters. If it is not an actual hash value, specify a hash_type of "None".
"hash_type"	The hash algorithm for determining the file hash (MD5, SHA-1, or SHA-256). None indicates that the file hash is not an actual hash value.
"severity"	(Required) The threat severity level (Low, Medium, High, or Critical).
"file_name"	(Required) The file name of the threat.
"file_size"	The size of the threat file, in bytes. -1 indicates that no file size was provided by the third-party source or that file_size is not present in the "ioc" response.
"platform"	The operating system (OS) for which the threat intelligence system reported the threat. All indicates that the threat was reported for all operating systems.
"cnc"	The CnC address (All operating systems). Known IP address that is used by specific malicious applications as a dial-home address.
"dns"	The DNS query (All operating systems). Known URL that is used by specific malicious applications as a dial-home address.
"file_exists"	The file exists (Windows only). Known specific file that is used by a malicious process.
"process"	The process (Windows only). Known malicious process.
"mutex"	The mutex (Windows only). Known OS mutex that can indicate a malicious process.

Field	Description
"registry"	The registry key (Windows only). Known OS registry values that can indicate a malicious process.
"service"	The service (Windows only), such as a Windows startup or scheduled system service. Known malicious service or a service that can be used by malicious factors.

Field data type requirement

The fields and types for the IOC dictionary are as follows:

Field	Type
"name"	String.
"date"	Integer (epoch time in milliseconds).
"hash"	String.
"hash_type"	String. The valid values for hash_type are: "md5", "sha1", and "sha256" (case-insensitive), corresponding to MD5, SHA-1, and SHA-256 hash algorithms. If the hash_type is different than the valid values, it is set to "None" to indicate that the hash is not an actual hash value.
"severity"	String. The valid values are: "low", "medium", "high", and "critical" (case-insensitive). If severity is not present or if its value is not valid, IOCs will not be sent.
"file_name"	String.
"file_size"	Integer. If file_size is not specified, the default value is -1.
"platform"	String. If platform is not specified, the default value is "None".
"cnc"	String (for a single IP address) or list of String (for multiple IP addresses).
"dns"	String (for a single query) or list of String (for multiple queries).

Field	Type
“file_exists”	Dictionary containing two keys: "file_name" and "file_path". Each key is a String or a List of String. To add a single file, specify a String value for each key. To add multiple files, specify a List of String for each key. The number of elements in the lists should match, otherwise, the file_exists data will be discarded. The values at the same index of the two lists form complete data for a “file_exists”.
“process”	Dictionary containing three keys: "process_name", "process_hash”, and "process_hash_type". Each key is a String or a List of String. To add a single process, specify a String value for each key. To add multiple processes, specify a List of String for each key. The number of elements in the lists should match, otherwise, the process data will be discarded. The values at the same index of the two lists form complete data for a “process”.
“mutex”	String (for a single mutex) or list of String (for multiple mutexes).
“registry”	Dictionary containing two keys: "registry_element" and "registry_data”. Each key is a String or a List of String. To add a single registry, specify a String value for each key. To add multiple registries, specify a List of String for each key. The number of elements in the lists should match, otherwise, the registry data will be discarded. The values at the same index of the two lists form complete data for a “registry”.
“service”	String (for a single service) or list of String (for multiple services).

Use the Connect web service

This topic describes the Connect web service.

The Connect web service APIs provide an interface to access eyeExtend Connect functions. Using the Connect RESTful APIs, a third-party vendor can get or update host information. You can query and update the properties defined in a Connect app through Connect RESTful APIs in a script or through Swagger or curl commands.

To use the Connect web service to invoke an API:

1. Enable the web service in the system.conf file for the app. See Define Name, Version, and Author in system.conf.
2. Enable the property values that you want to update through the APIs in the property.conf file for the app. See Define Properties in property.conf.

3. Configure the user credentials in the Web Service Authentication tab in the Connect Plugin.. See Web service authentication tab in Connect pane.
4. Obtain a JWT token. The RESTful API uses the user credentials to validate the app or apps. The JWT token can be used by two apps.

   The JWT token, with the Bearer token format, is then used for subsequent API requests to authenticate with the Connect Plugin.

5. Call RESTful APIs to get or update host information.

   You can use Swagger to obtain a JWT token and then use the Authorize button in Swagger to include the JWT token with the Bearer token format. See Access Swagger User Interface, Obtain JWT Token from Swagger, and Set Authorization in Swagger.

   You can also use curl or POSTMAN to call the APIs, but then you will need to put the JWT token with the Bearer token format in the API request Authorization header. See Curl Examples.

Access swagger user interface

Swagger (OpenAPI) documents the Connect web API and lets you run the RESTful APIs by using the Try it out button. You can browse Swagger to look at the available APIs, their parameters, response definitions, and data model, as well as to learn how the APIs work with request and response. Swagger provides a user-friendly interface to invoke APIs.

You do not need to authenticate to access the Connect web service Swagger user interface (UI), however, you will need to authorize to run any API via Swagger.

To view a static version of the Swagger UI, see Appendix C: Swagger User Interface.

To access the live version of Swagger, in a supported browser, substitute the hostname or IP address of your Enterprise Manager (EM) in the following URL:

```
https://<EM hostname or IP address>/connect/swagger-ui/index.html#/
```

Swagger is supported in the latest versions of Chrome, Safari, Firefox, and Edge available at the time of this release.

About the Connect APIs

The following Connect APIs are available:

- POST /connect/v1/authentication/token: Use this API to get a JWT token to authenticate API requests. The required parameters are "username", "password", and "app_name". To get a token for one app:

```
{
    "username":"username",
    "password":"password",
    "app_name":"sampleapp",
```

```
    "expiration":"15"
}
```

To get a token for two apps, use a comma separated list:

```
"app_name":"sampleapp, cylance",
```

- GET /connect/v1: Use this API to test the connection to Connect. There are no parameters.
- GET /connect/v1/hosts/{id}: Use this API to get properties for a single host. To GET host properties, the endpoint called must be the EM or managing appliance. The required parameters are IP address or MAC address. Properties are optional. If no properties are specified, all the properties in the app are returned. An example of a single property is: connect_sampleapp_is_safe. An example of multiple properties is: connect_sampleapp_is_safe, connect_sampleapp_last_logged_in_user (use a comma separated list).

  Note:
  If you specified two apps when you obtained the JWT token, you can get properties from both apps.
- POST /connect/v1/hosts/: Use this API to update one or more properties for a host. Each of the properties to be updated must have "web_enable" set to true. The required parameters are an IP or MAC address as well as the properties to be updated. The parameters must be in the body of the POST request. For this release, only one host update per request is supported. If both the IP and MAC addresses are provided in the body, the MAC address takes precedence over the IP address. If the IP address is different from the existing IP address, the IP address of the host is updated to the IP address provided in the request body. The following is a sample request body:

```
{

"ipv6":

["<ipv6>","<ipv6 2>","<ipv6 3>"],

"ip":"<ipv4 address>",

"properties":{

}
```

  Note:
  If you specified two apps when you obtained the JWT token, you can post properties to both apps.

You can try out the APIs in Swagger by using the Try it out button. You will need to set up the Authentication in Swagger before calling the APIs. See Obtain JWT Token from Swagger and then Set Authorization in Swagger.

After setting up authentication in Swagger, you can edit the required fields in the API request and then select Execute. You can review the response codes and values in Swagger. See API Response Table for a response code reference.

Additional API details

The following are some details for the APIs:

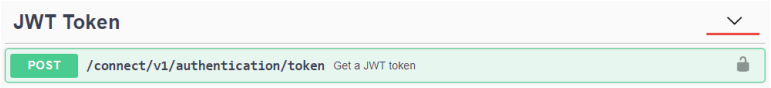
- HTTPS is required for requests.
- All API requests need authentication:
  - To obtain the JWT token, the username and password are used in the request.
  - For subsequent API requests, the JWT token in Bearer format is used in the request.
- The bearer token format is the word Bearer, followed by a single space and the JWT token.
- The context type accepted in the request and response body is application/json.
- The response structure is always in JSON format.
- The JWT token default expiration is 15 minutes. The range is from 1 minute to 1440 minutes. A shorter expiration period makes a token more secure.

Obtain JWT token from Swagger

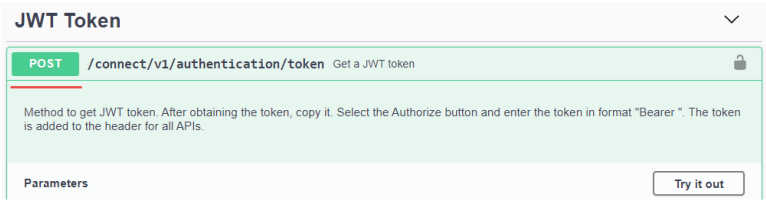
Use Swagger to obtain a JWT token.

To obtain a JWT token from Swagger:

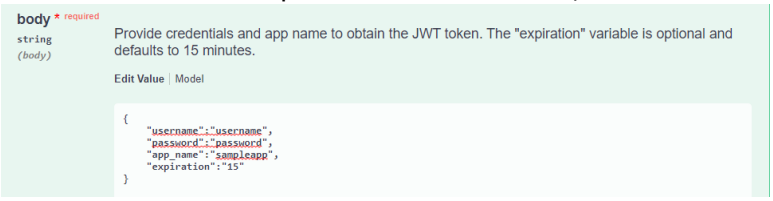
1. Go to Swagger, substituting the hostname or IP address of your Enterprise Manager in the following URL: `https:// <EM hostname or IP address>/connect/swagger-ui/index.html#/`
2. Expand the JWT Token.



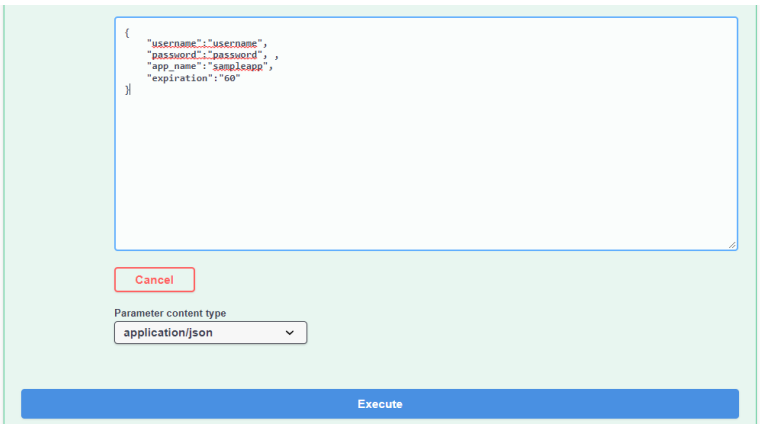
3. Select POST and then select Try it out.



4. In the sample body in JSON format, enter the required values from your Connect app for username, password, and app_name. You can enter a value for expiration, but it is optional. The default expiration is 15 minutes, the maximum is 1440 minutes.

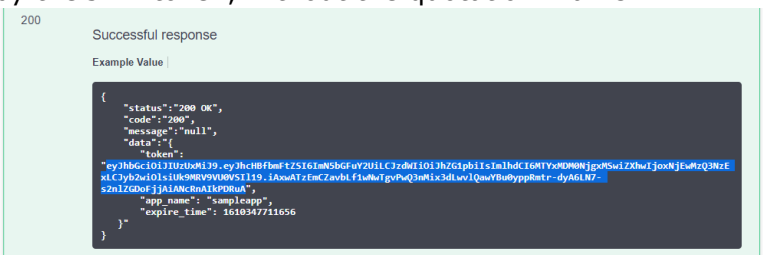


5. After entering your values, select Execute.



6. Scroll to the Successful response (Code 200).

7. Copy the JWT token, without the quotation marks.



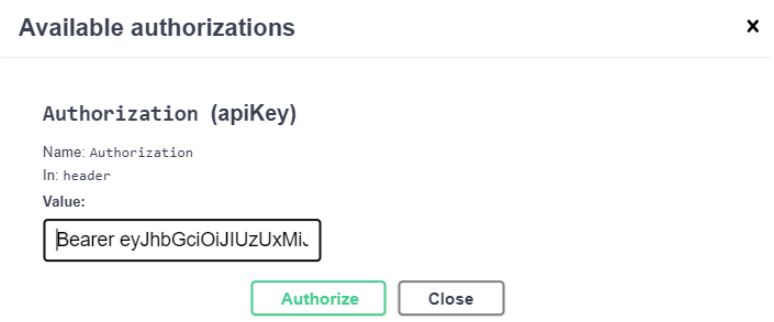
Set authorization in Swagger

Use Swagger to set authorization to run other APIs.

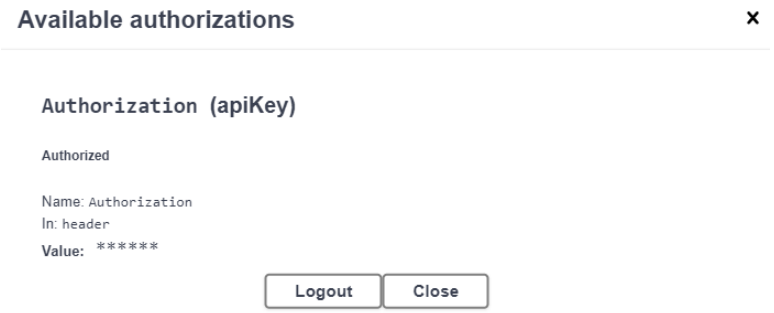
To set authorization in Swagger:

1. Scroll to the Authorize button and select it.

2. Type the word Bearer, a single space, and then paste the JWT token into the Value field.



3. Select Authorize.



4. Select Close.

With authorization set, you can now run other APIs in Swagger. For example, go to GET /connect/v1, select Try it out, and then select Execute and view the response.

Curl examples

The following is a curl example to get the JWT token. Substitute your EM hostname or IP address for <EM hostname or IP>.

```
curl -X POST "https://<EM hostname or IP>/connect/v1/authentication/token"
-H "accept: application/json" -H "Content-Type: application/json" -d
"{\"username\":\"username\", \"password\":\"password\",
\"app_name\":\"sampleapp\", \"expiration\":\"15\"}"
```

The following is a curl example to use the JWT token in a request. Substitute your EM hostname or IP address for <EM hostname or IP>.

```
curl -X GET "https://<EM hostname or IP>/connect/v1" -H "accept:
application/json" -k -H "Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJhcHBfbmFtZSI6ImN5bGFuY2UiLCJzdWIiOiJjaGFybGllIiwiaW
F0IjoxNjEzNzcxNDg3LCJleHAiOjE2MTM3NzIzODcsInJvbCI6WyJST0xFX1VTRVIiXX0.aXmhX
Y4B_lV5RSDhUphuOcH6Y8QSsxB47FCWwDNXo-
```

```
SVh2ssK8nVBTqE-5UpziFyYFuagpu8KLpZlOSP4fLYKQ"
{"status":"OK","code":200,"message":null,"data":"Connect"}
```

API Response Structure

API responses are always in JSON format.

Successful response

A successful response has the following structure:

- "status": The HTTP status, in string, such as OK.
- "code": The HTTP status code, in integer.
- "message": A message if there is any, in string. Otherwise, null.
- "data": The data returned, in JSON format.

The following is an example of a successful response:

```
{
  "status": "OK",
  "code": 200,
  "message": null,
  "data": {
    "token":
"eyJhbGciOiJIUzUxMiJ9.eyJhcHBfbmFtZSI6ImN5bGFuY2UiLCJzdWIiOiJjaGFybGllIiwia
WF0IjoxNjEwMTU2Nzg0LCJleHAiOjE2MTAxNTc2ODQsInJvbCI6WyJST0xFX1VTRVIiXX0.BufP
f-gtgVT8zWMxUqnQ-BxdKWm2HUDv8dEnukIojqqd_G0voscweZ48Wod4PN17H3et-eZ5oX2-
NJ6aU5GyFw",
    "app_name": "sampleapp",
    "expire_time": 1610157684367
  }
}
```

Failed response

A failed response has the following structure, which is similar to the successful response, but there is no data parameter:

- "status": The HTTP status, in string, such as UNAUTHORIZED, or other error.
- "code": The HTTP status code, in integer.
- "message": An error message if there is any, in string. Otherwise, null.

See API Response Table for a response code reference.

The following is an example of a failed response:

```
{
  "status": "UNAUTHORIZED",
  "code": 401,
  "message": "Bad credentials"
}
```

API response table

The following table lists API response error messages.

Condition	Status	Code	Message
GET https:// EM_IP/ connect/v1			
Succeed	OK	200	null
Missing Authorization header	UNAUTHORIZED	401	Cannot find authorization header info.
Not using Bearer token	UNAUTHORIZED	401	JWT token does not begin with Bearer format.
Invalid token format	UNAUTHORIZED	401	Request to parse invalid JWT token failed: <exception message>
Expired token	UNAUTHORIZED	401	Request to parse expired JWT token failed: <exception message>
Invalid token signature	UNAUTHORIZED	401	Request to parse JWT token with invalid signature: <token> failed: <exception message>
unsupported token format	UNAUTHORIZED	401	Request to parse unsupported JWT token failed: <exception message>
Server failed to respond	INTERNAL_ SERVER_ERROR	500	Message varies depending on the failure.

Condition	Status	Code	Message
POST https:// EM_IP/ connect/v1/ token			
Succeed	OK	200	null
Succeed	MULTI_STATUS	207	The following apps are not authorized: <app name>
Using Get	BAD_REQUEST	400	Use POST instead of GET.
Missing 'username' in body	BAD_REQUEST	400	Username is missing.
Missing 'password' in body	BAD_REQUEST	400	Password is missing.
Missing 'app_name' in body	BAD_REQUEST	400	App name is missing.
Invalid JSON body	BAD_REQUEST	400	JsonParseException message. For example: Unexpected character ('\"' (code 34)): was expecting comma to separate Object entries\n at [Source: (forescout.plugin.connect_module.web.security.jwt.CachedBodyServletInputStrean line: 4, column: 6]"
Credential not correct	UNAUTHORIZED	401	Bad credentials
GET https:// <EM_IP>/ connect/v1/ hosts/<id>			
Succeed	OK	200	null

Condition	Status	Code	Message
Missing Authorization header	UNAUTHORIZED	401	Cannot find authorization header info.
Not using Bearer token	UNAUTHORIZED	401	JWT token does not begin with Bearer format.
Invalid token format	UNAUTHORIZED	401	Request to parse invalid JWT token failed: <exception message>
Expired token	UNAUTHORIZED	401	Request to parse expired JWT token failed: <exception message>
Invalid token signature	UNAUTHORIZED	401	Request to parse JWT token with invalid signature: <token> failed: <exception message>
unsupported token format	UNAUTHORIZED	401	Request to parse unsupported JWT token failed: <exception message>
Invalid mac or ip address	BAD_REQUEST	400	Host id <id> is not in correct format. Provide IP or MAC address.
Host not exist	NOT_FOUND	404	Host not found by <IP/IPv6/MAC> address <id>
Unable to get properties' names for the given app	INTERNAL_ SERVER_ERROR	500	Unable to get property names for apps: [app_name]
Exception when get host properties on the infra side	INTERNAL_ SERVER_ERROR	500	Unable to get host properties. Exception: <exception message>
POST https:// EM_IP/			

Condition	Status	Code	Message
connect/v1/hosts			
Succeed	OK	200	null
Invalid mac or ip address	BAD_REQUEST	400	MAC address or IP address is not in a correct format.
Missing mac and ip	BAD_REQUEST	400	Missing IP and MAC address in the request body. At least one of them is required.
Request body is not a valid json	BAD_REQUEST	400	Request body is not in a valid JSON format.
Updating a property that does not have web_enable set	BAD_REQUEST	400	Failed to update a property that does not have web_enable set: <property name>
value for the property is in wrong type	BAD_REQUEST	400	Value for <property name> is not in correct format, expecting <right type>
property doesn't exist	BAD_REQUEST	400	Failed to update an undefined property: <property name>
update property belongs to other app	FORBIDDEN	403	Failed to update a property that is not authorized in the JWT token.
unknown exception when updating property	INTERNAL_SERVER_ERROR	500	Failed to resolve host properties. Exception: <exception message>

Condition	Status	Code	Message
plugin is not responding	INTERNAL_SERVER_ERROR	500	Plugin is not responding.
unknown error	INTERNAL_SERVER_ERROR	500	Failed to update host properties.
Failed to send request to plugin	INTERNAL_SERVER_ERROR	500	Failed to submit request to plugin.

Connect web service logs

Tomcat is shipped with the Forescout Platform as a web server.

The Connect web service logs are located in the following path:

`/usr/local/tomcat/webapps/logs/connect-tomcat.log`

The Tomcat-related logs are located in the following path:

`/usr/local/tomcat/logs`

Change log level

There are two methods to change the log level.

The first method is to modify the logback-spring.xml file located in the following path:

`/usr/local/forescout/webapps/connect/WEB-INF/classes/logback-spring.xml`

In the .xml file, change the following variable to the desired debug level: ERROR, WARN, INFO, DEBUG, or TRACE.

`logging.level.forescout.plugin.connect_module.web=INFO`

After changing the log level, restart the Tomcat web server for the log level to take effect.

The second method to change the log level is through the Spring Boot logger APIs. This does not require a web server restart. You must include the Bearer token in the API.

The following is a curl example to get the connect web service log. Substitute your hostname or IP address for `<hostname or IP>`.

```
curl -H "Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJhcHBfbmFtZSI6ImN5bGFuY2UiLCJzdWIiOiJjaGFybGllIiwiaW
F0IjoxNjExMTA3NzIwLCJleHAiOjE2MTExMDg2MjAsInJvbCI6WyJST0xFX1VTRVIiXX0.UjVWD
```

```
feBqN6jGH-
w_nolvyKYIEVebRgHeQUXSdbdCKpmGpiJFytJNulYEcNo4EnRn4HQFyx1PNfOa1ONc-JruQ"
-k -X GET https://<hostname or IP>/connect/actuator/loggers/
forescout.plugin.connect_module.web
```

The following is a curl example to set the connect web service log to the debug level. Substitute your hostname or IP address for `<hostname or IP>`.

```
curl -k -X POST -H "Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJhcHBfbmFtZSI6ImN5bGFuY2UiLCJzdWIiOiJjaGFybGllIiwiaW
F0IjoxNjExMTA3NzIwLCJleHAiOjE2MTExMDg2MjAsInJvbCI6WyJST0xFX1VTRVIiXX0.UjVWD
feBqN6jGH-
w_nolvyKYIEVebRgHeQUXSdbdCKpmGpiJFytJNulYEcNo4EnRn4HQFyx1PNfOa1ONc-JruQ"
-d '{"configuredLevel": "DEBUG"}' https://<hostname or IP>/connect/actuator/
loggers/forescout.plugin.connect_module.web
```

Deploy an app with Connect

This topic describes how to deploy an app with Connect.

This topic 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 topics:

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

Download a Connect app from GitHub

This topic describes downloading a Connect App from GitHub.

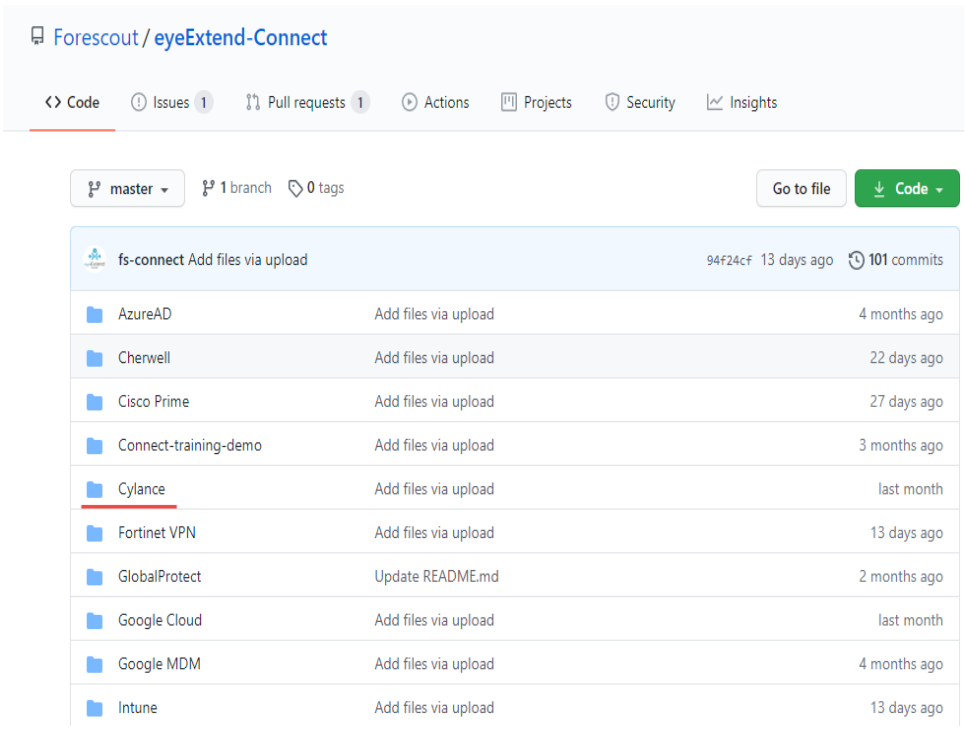
The eyeExtend Connect Apps are located on the Forescout eyeExtend Connect GitHub page as follows:

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

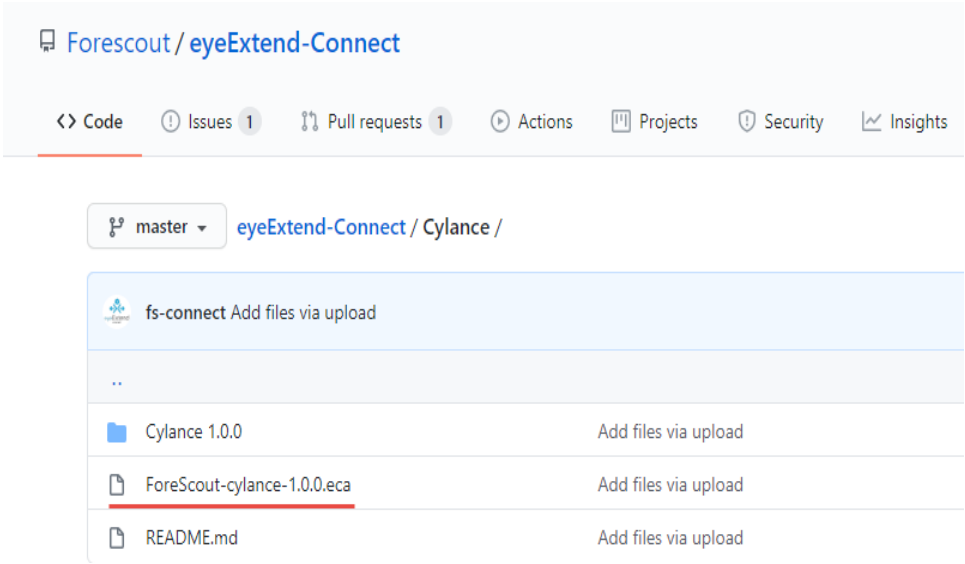
Apps signed by Forescout are in .eca format and are downloaded from GitHub as a .eca image file.

To download a .eca image file for a Connect App from GitHub:

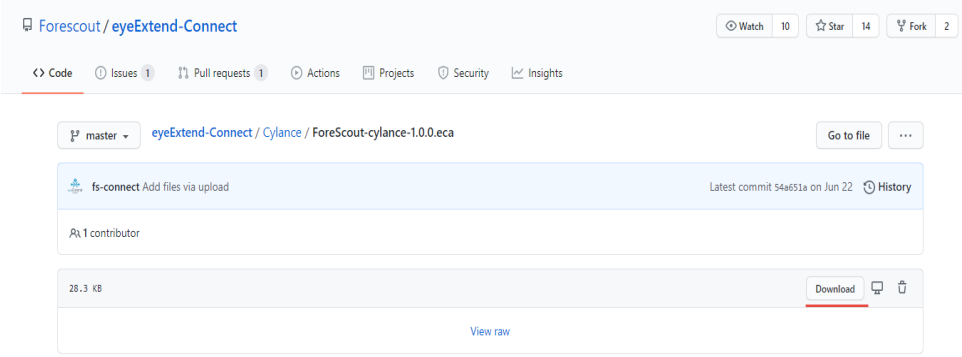
1. Go to the Forescout eyeExtend Connect GitHub page.
2. Select an app, for example, Cylance.



3. Select the .eca image file for that app.



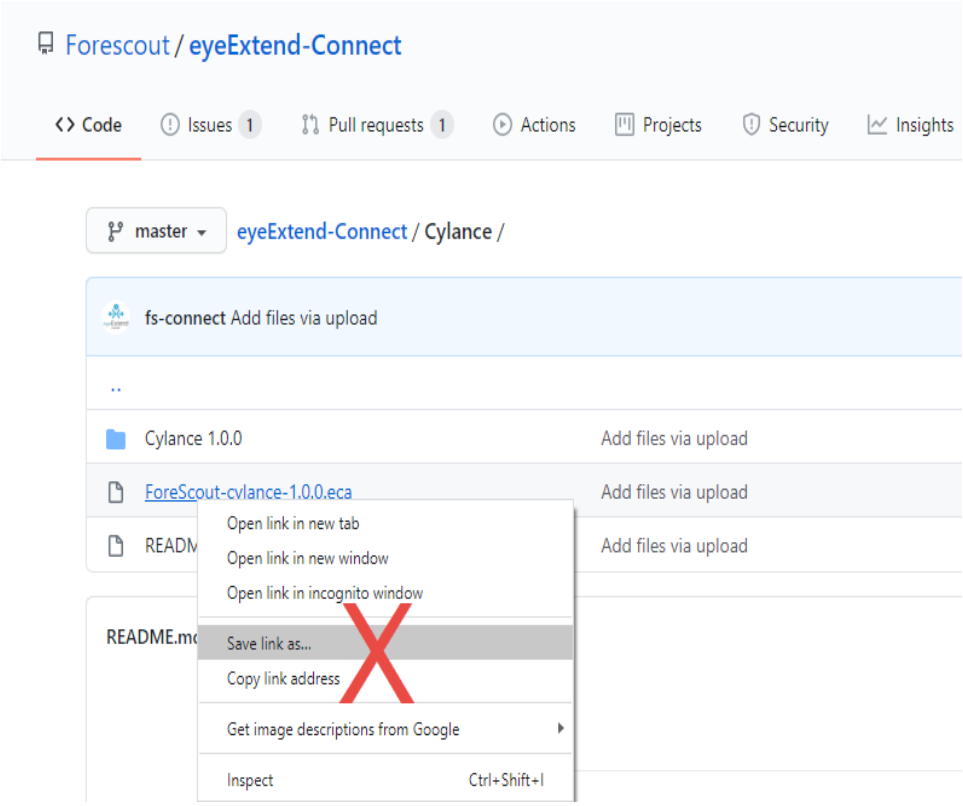
4. Select Download.



App download issue

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

For example, it is not correct 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.



Install Connect plugin

This topic describes how to install the Connect Plugin.

To install the module:

1. Navigate to the Downloads page on the Forescout Customer Support Portal
2. Download the `.fpi` file for the component
3. Save the file to the machine that has the console installed
4. Log in to the console and select Tools - Options from the menu
5. Select Modules - Install. The Open dialog box appears.
6. Find and select the saved component `.fpi` file
7. Select Install, which will open the installation window
8. Read the License Agreement and select I agree to the License Agreement, then Install. The installation begins immediately after selecting Install and cannot be interrupted or canceled. In modules with multiple components, components are processed serially.
9. When the installation completes, select Close to close the window. The installed component (module or plugin) is displayed in the Modules pane.
10. Start the plugin by select Tools - Options - Modules, then right-click on the component name in the Modules pane.
11. In the menu that appears, select Start

Connect Add-On mode and web service install

The Connect web service installs and deploys from the Connect plugin. It may take a minute or two for the web service to fully run and accept web requests after deployment. Once you uninstall the Connect plugin, the Connect web service uninstalls.

Connect Add-On optional mode

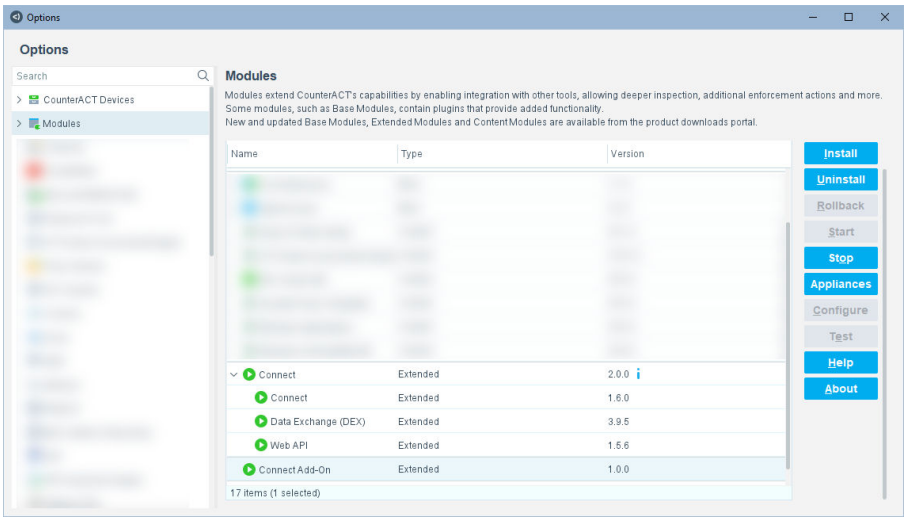
The Connect Add-On module is an optional module that's dependent on the installation of the Connect plugin. Connect Add-On provides licensing of an additional 20 apps, two of which are included with the Connect plugin license (totalling 22 apps).

If you need to run more than two apps, you must 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.

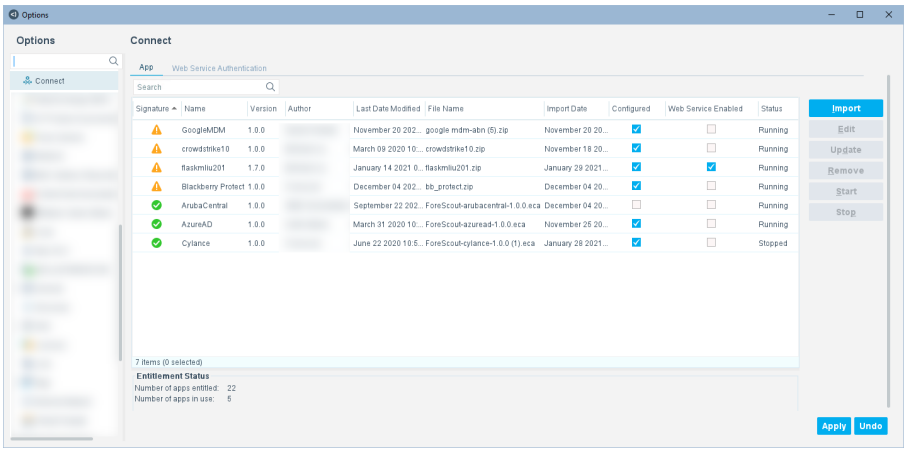
Refer to the Connect Add-On Module How-to Guide for more information.

Connect Add-On installed

The Connect plugin and the Connect Add-On module can be installed from the console by selecting Options - Modules. The Connect Add-On module doesn't require any configuration, nor does it have to be running.



The Entitlement Status will then display in the Connect pane:

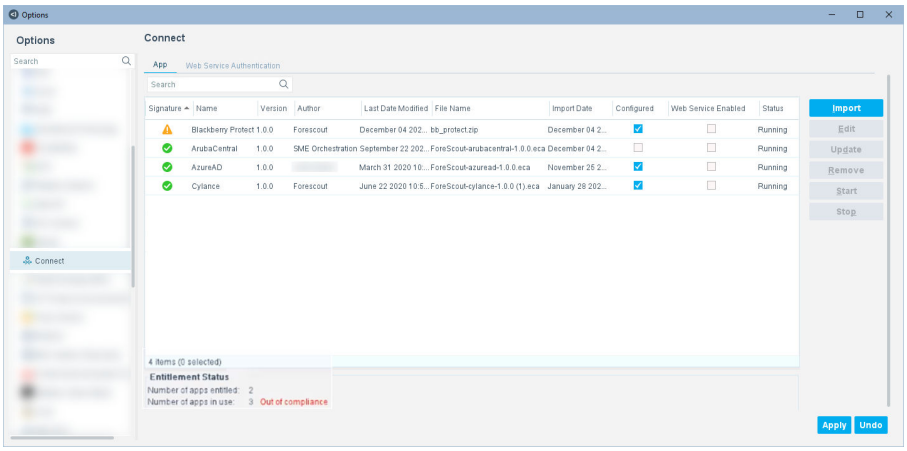


If an app isn't in a Configured or the Running state, it's not counted as *in use*. The example shows seven apps, but only five are both Configured and Running (one is not Configured and one is Stopped), so the count of the number of apps in use is five.

The entitlement status will be in compliance as long as the number of apps doesn't exceed 22.

Connect Add-On uninstalled

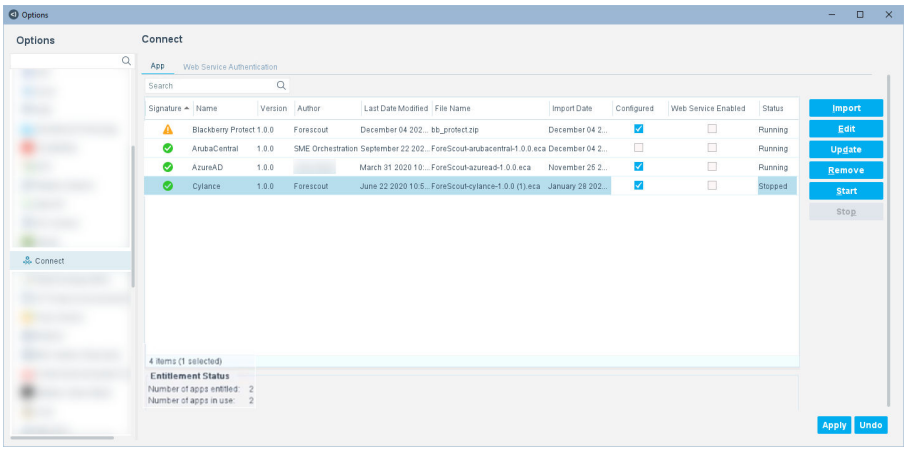
If you don't have the Connect Add-On module installed, you are entitled to two apps with the Connect plugin license. The Entitlement Status is displayed in the Connect pane:



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 or Running state, it isn't 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*:



Connect user interface details

This topic provides details of the user interface.

See the following topics:

- Connect pane details
- Web service authentication tab in Connect pane
- System description dialog box details

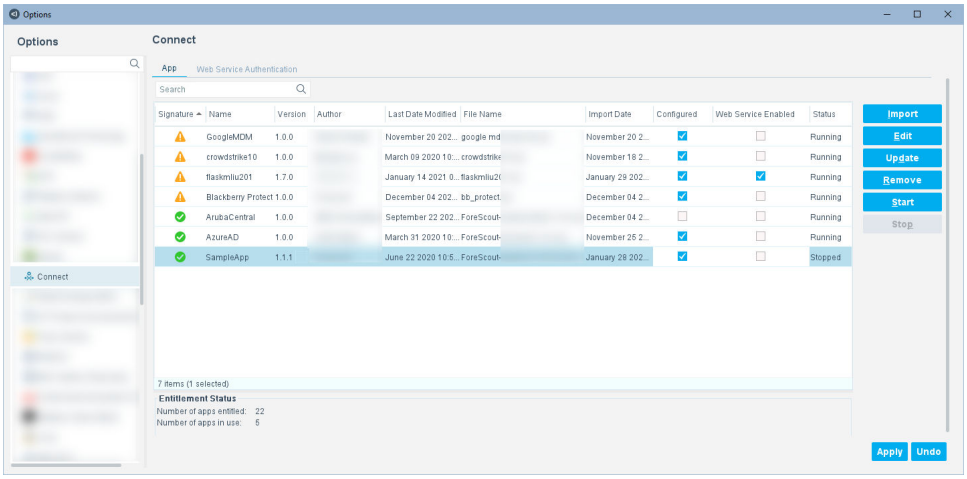
- Configure policy templates in Connect

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.

There are two tabs on the Connect pane:

- The App tab, which manages apps
- The Web Service Authentication tab, which configures authentication for the Connect web service. To learn more, see Web service authentication tab in Connect pane.



Columns in Connect pane

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

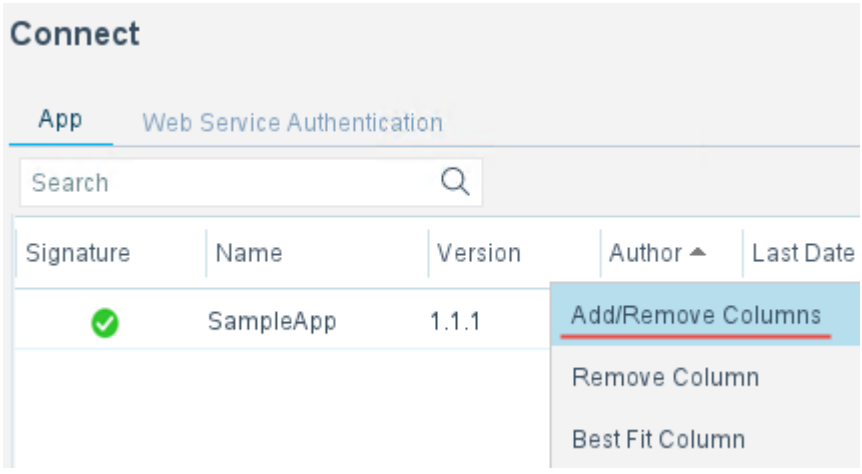
Signature	The signature of the imported app. A green check mark indicates that the signature is valid, while an orange caution sign indicates it's invalid or missing. Apps from Forescout are signed after their creation, to ensure their authenticity and integrity. To learn more, see Import a signed app.
Name	The name of the third-party integration app 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
Import Date	The date the app was imported
Configured	The configuration flag, which has a check mark if the system description is configured. If an app has been imported, but not configured, there will not be a check mark in this column. To configure the system description, see Add a system description.
Web Service Enabled	The web service flag has a check mark if the Connect web service is enabled
Status	The status of the app. The valid values are Running and Stopped.

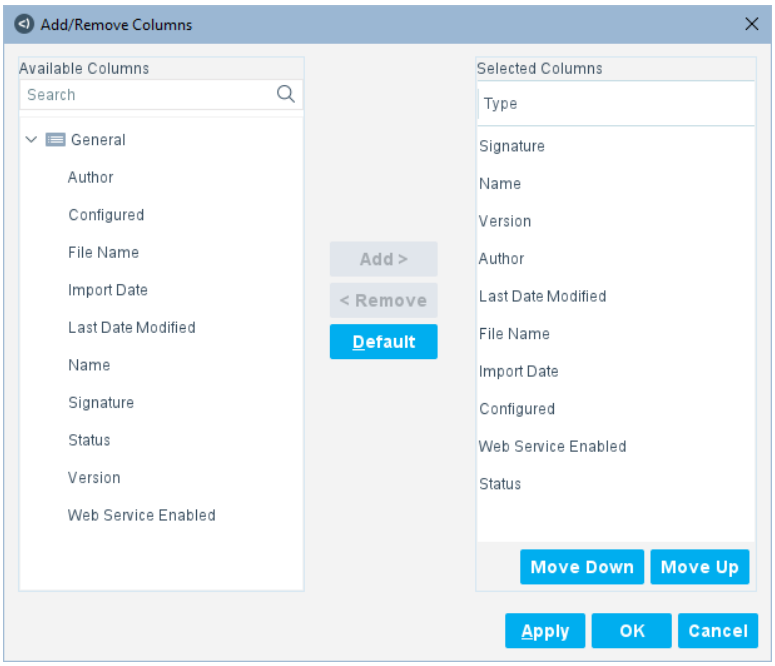
At the bottom of the Connect pane is the Entitlement Status:

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. If there's 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 displays:



To add, remove, or reorder the columns on the Connect pane, select Add/Remove Columns. You can then 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

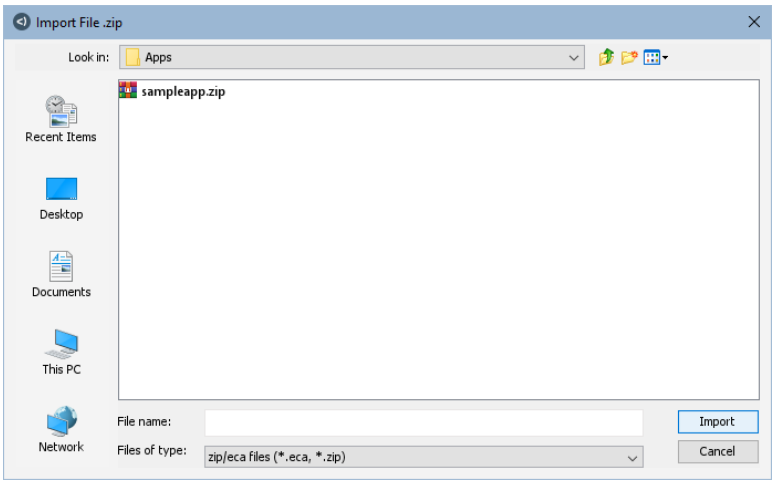
This topic describes buttons (actions) within the Connect pane.

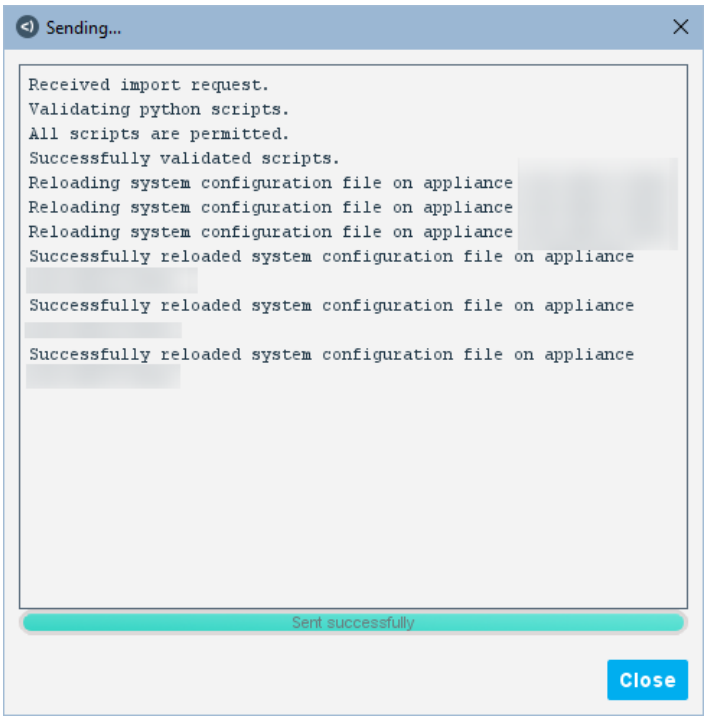
There are several buttons on the Connect pane for apps:

- Import an app or signed app
- Update
- Edit
- Remove an app or app property added to discovery
- Start
- Stop
- plus Apply and Upgrade for changes made in the pane

Import an app

Select the Import button to import apps into Connect, either in zip or eca format. Messages display as the app is imported:

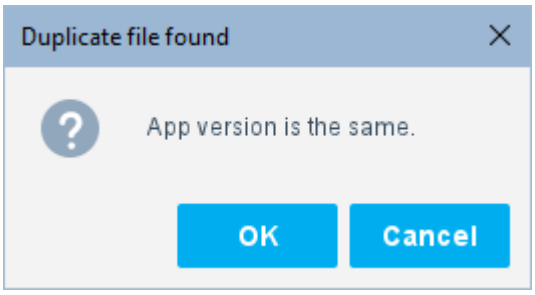




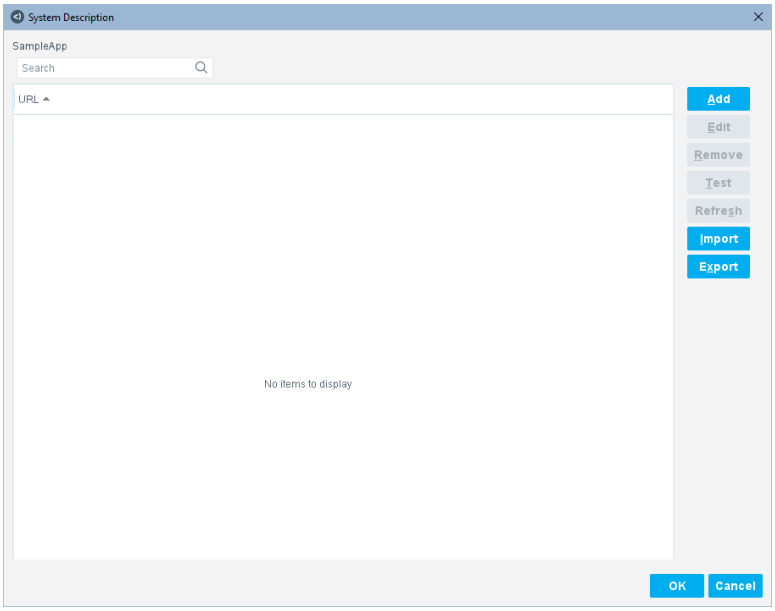
For a successful import, a message will display at the bottom of the Sending dialog box. If the app hasn't imported successfully, error messages are displayed in the Sending dialog box.

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:



In this example, there is one column, URL, which has been defined in the system description (in the system.conf file):



The maximum number of apps that can be imported is 22. If a device hasn't been configured and you select OK in the System Description dialog box, a warning message will display.

To configure a system description, select Add.

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.

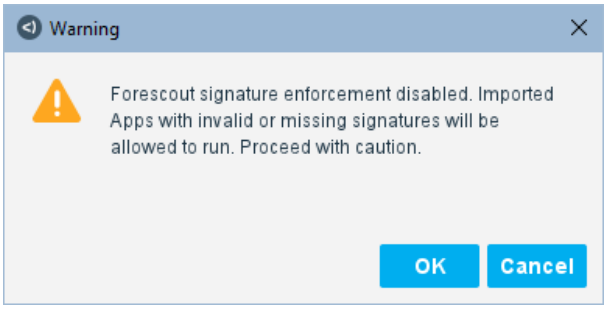
Note:
Unsigned apps might cause negative impacts on hosting CounterACT Appliances.

When you import an app, the signature of the app is checked for 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 will not import.

To import an app with an invalid signature, use the following command on the Enterprise Manager:

```
fstool allow_unsigned_connect_app_install true
```

This global command 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:



Update an app

If the system.conf and property.conf files have changed, you must upgrade the app. See Upgrade an app.

If you've made the following changes, you can use the Update button to update an app:

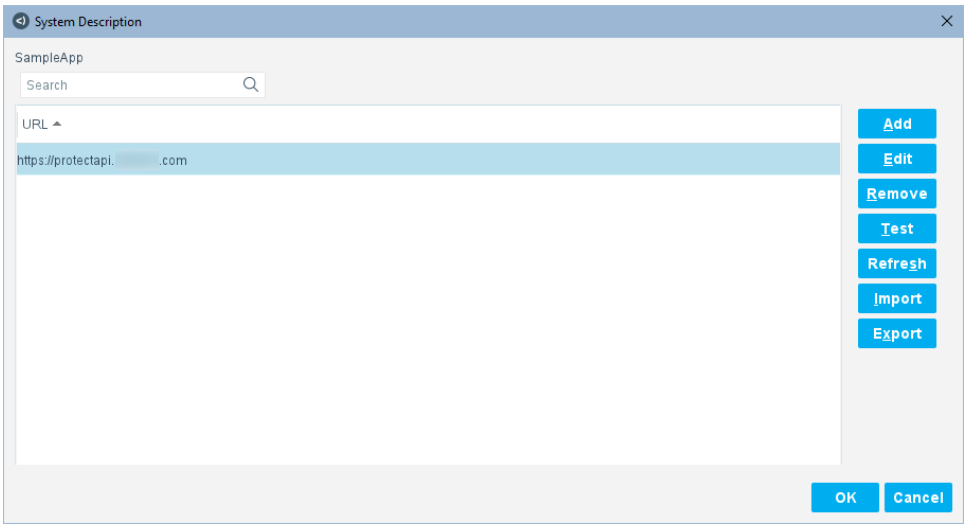
- "version" change only (to a higher or a lower version) in the system.conf file
- any content change in existing scripts

An error message displays 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

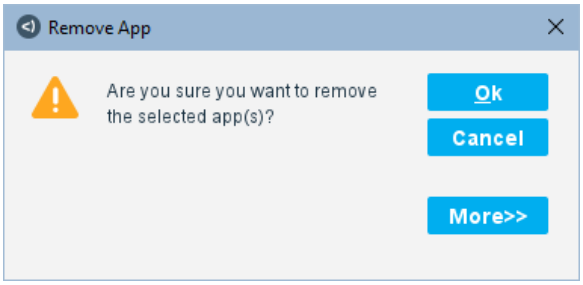
Select an existing app in the Connect pane and select Edia or double-click the existing app to open the system description for it:



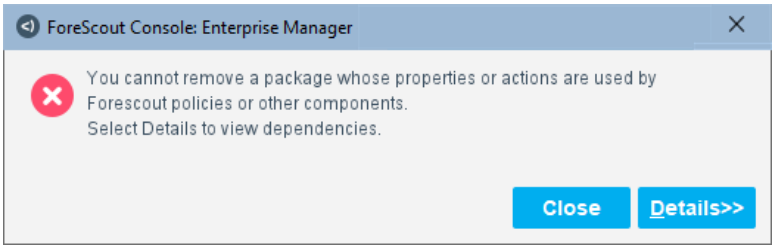
Remove an app

It's recommended to 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 select OK to confirm the removal:



Dependencies for properties and actions are checked before an app is removed. An error message is displayed if there are properties or actions configured in a policy when you try to remove the app:

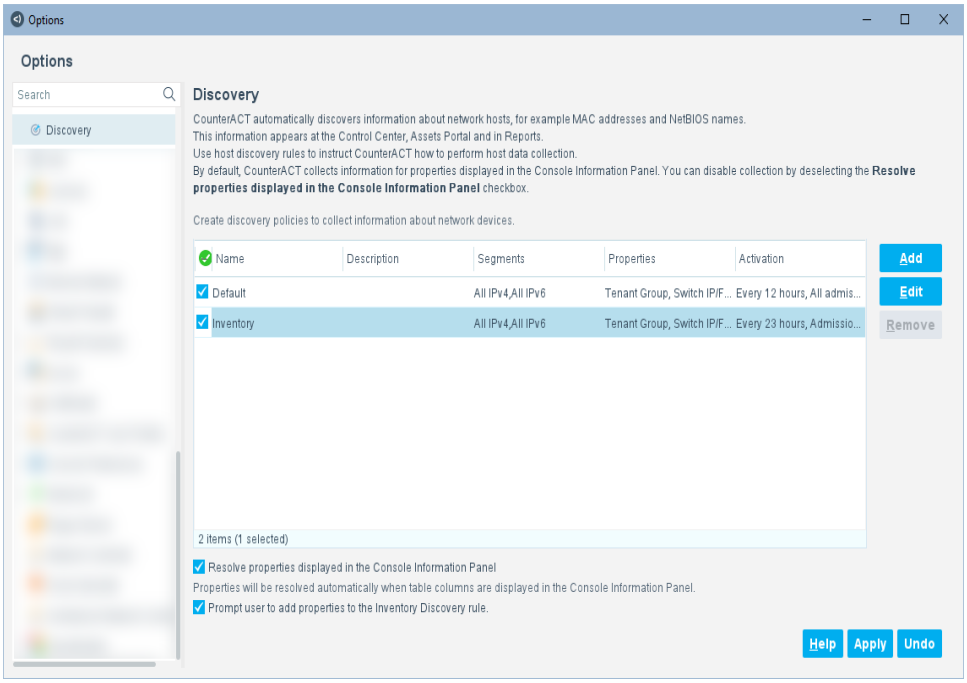


Select Details to view the specific properties and actions that are configured. You may have to remove the policy.

Select Close and then select Apply in the Connect pane.

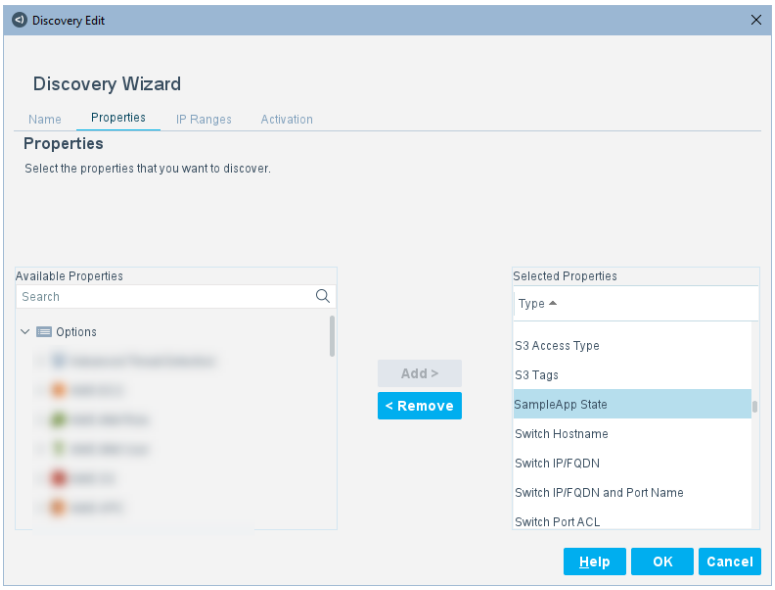
Remove an app property added to discovery

Before an app is removed, there is also a check to see that the properties are not referenced in Options - Discovery. For example, a property could have been added to Inventory:



If properties are referenced, an error message is displayed in Connect when you try to remove the app. Select Details to view the specific properties.

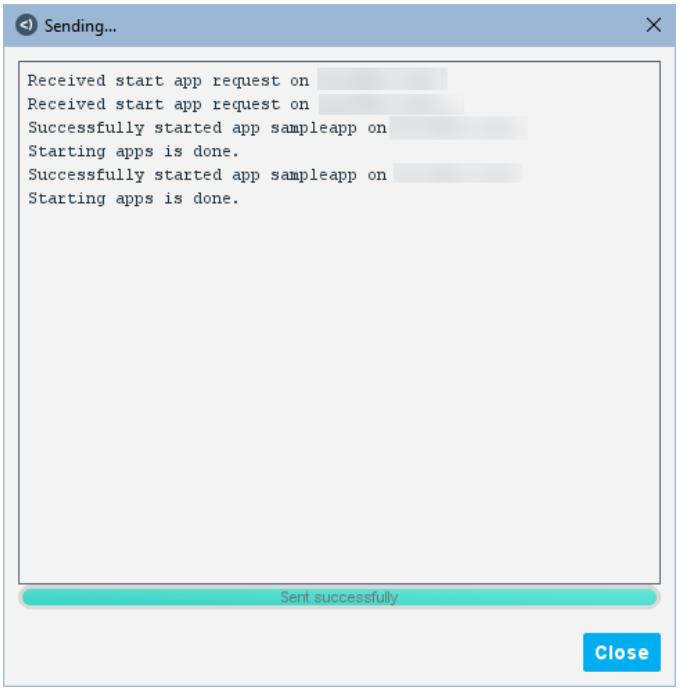
Remove the referenced properties before removing the Connect app. In the Discovery pane, select Edit and then select the Properties tab. In the *Discovery Wizard* dialog box, select the property on the right in Selected Properties and select Remove to move the property to the left to Available Properties:



Select OK and then select Apply in the Discovery 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):



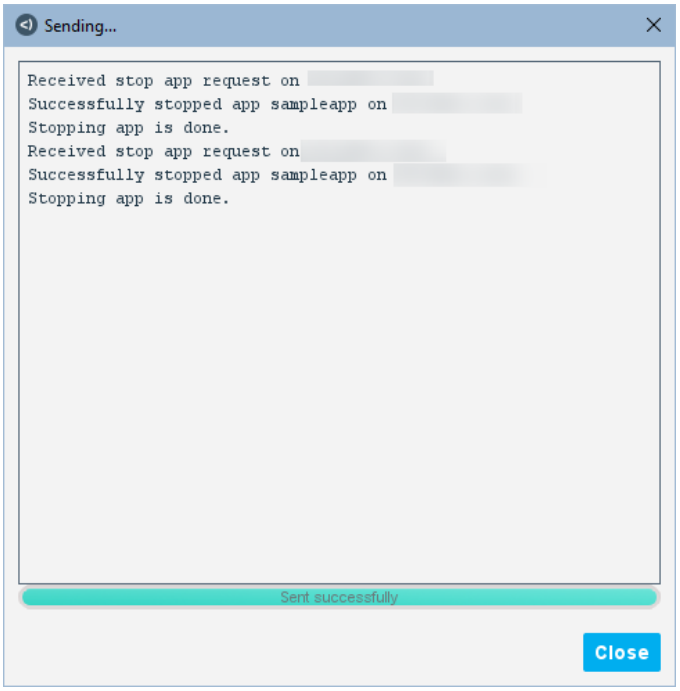
When a selected app is running, the Start button is disabled. After an app is started, a status of Running is displayed in the Connect pane. If an app is not selected, both the Start and Stop buttons are disabled.

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:

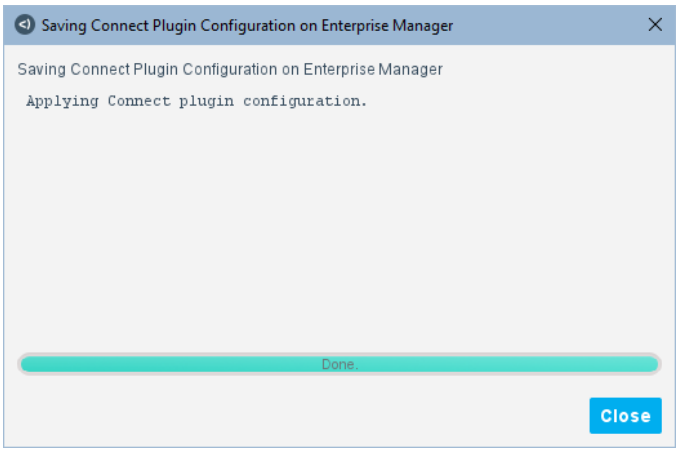


When a selected app is not running, the Stop button is disabled. After an app is stopped, a status of Stopped is displayed in the Connect pane. If an app is not selected, both the Start and Stop buttons are disabled.

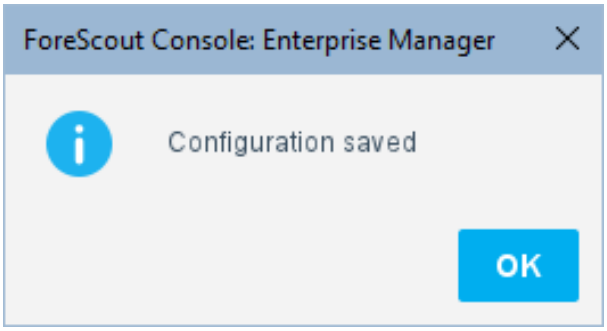
If the configuration has not been saved, select Apply and then select Stop.

Apply changes

Select the Apply button to save the changes to the configuration:



Select Close then OK to save:



Upgrade an app

To upgrade an app when the system.conf and property.conf files have changed

1. Select Remove then Import to add the newer app
2. Select Add for the system description

To learn more, see Remove an app, Import an app, and Add a system description.

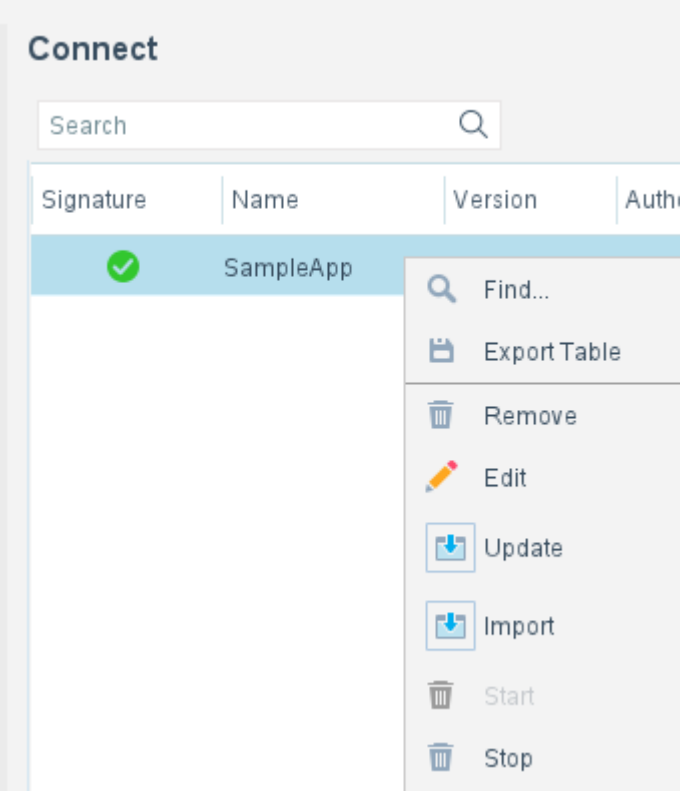
Since removing an app removes all configuration, you can also Export the system description before removing the app, then Import the app and Import the system description.

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. To learn more, see Update an app.

Connect pane menu

This topic describes the menu in the Connect pane, as well as several dialog boxes.

When you right-click an existing app in the Connect pane, a menu will appear with the following buttons: Remove, Edit, Update, Import, Start, and Stop. To learn more, see Buttons in Connect pane.

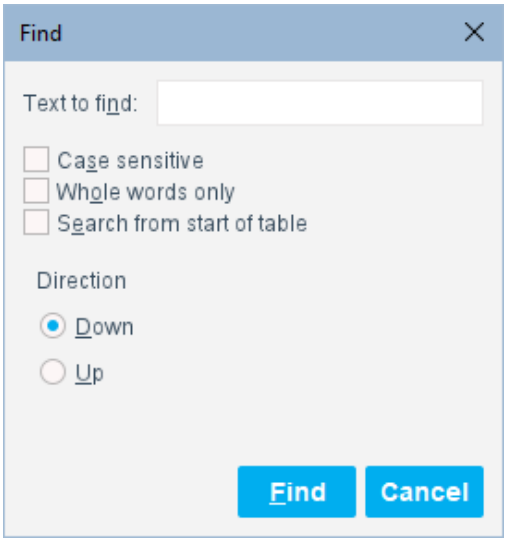


For Find and Export Table, see Find dialog box and Export table dialog box.

Find dialog box

To find a string:

1. Select Find:

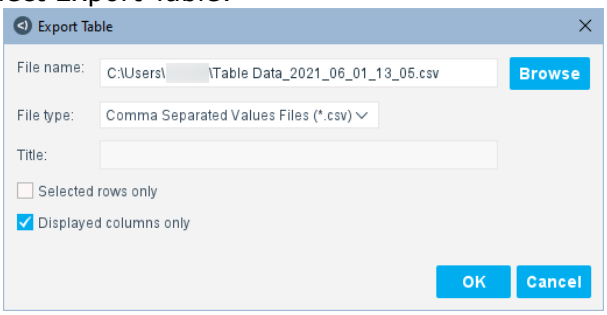


2. Enter the text to find, select check boxes or Direction, and then select Find

Export table dialog box

To export a table:

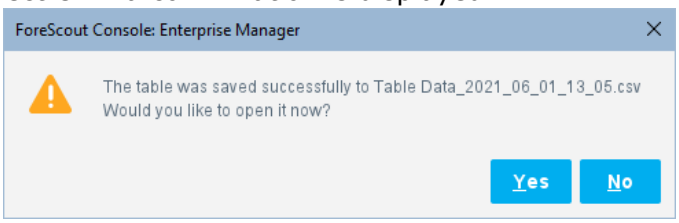
1. Select Export Table:



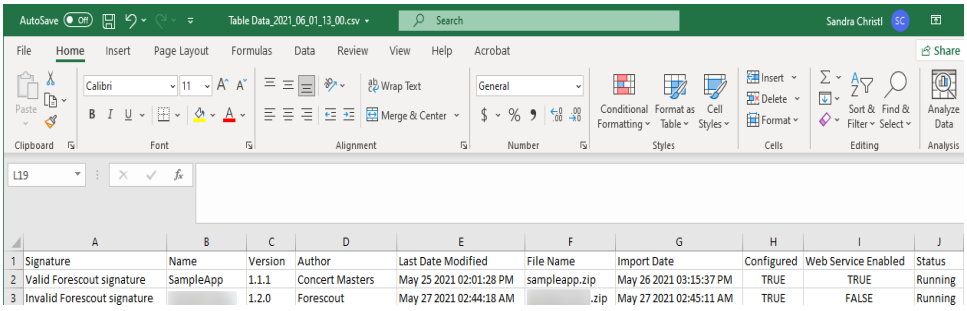
2. You can do the following:
   - Export the Selected rows only and/or the Displayed columns only
   - Change the File type to either Comma Separated Values Files (*.csv) or Acrobat Reader Files (*.pdf)
   - Change the folder location using Browse

     Note:

     You cannot enter anything in the Title field.
3. Select OK - a confirmation is displayed:



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_2021_06_01_13_00.csv-Excel:



Web service authentication tab in Connect pane

This topic describes the Web Service Authentication tab in the Connect pane.

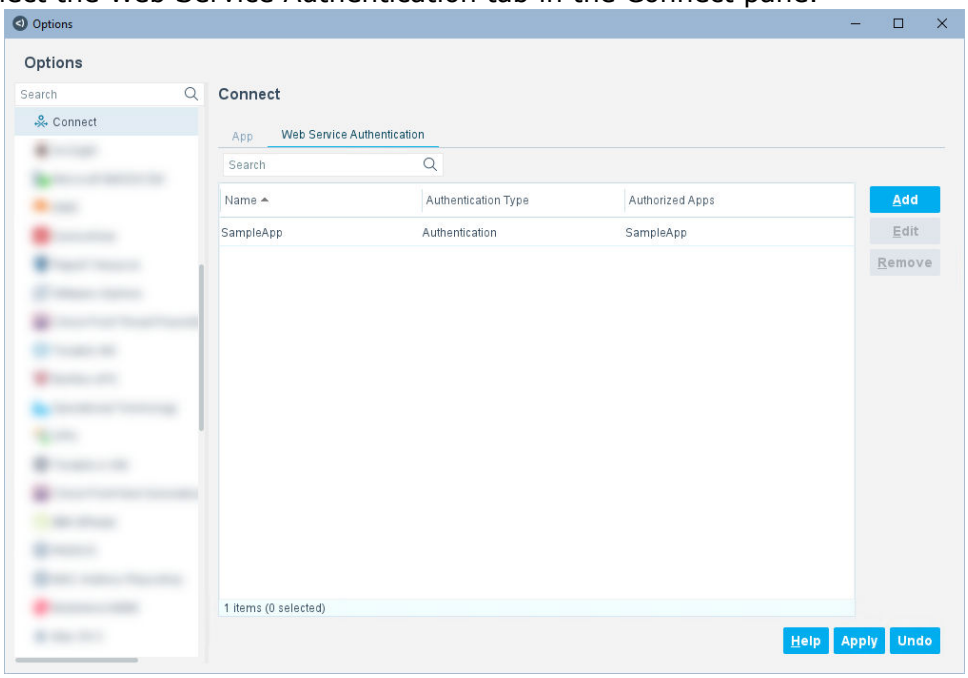
For authentication to take place, the configuration must include the user name and password used by the Connect web service APIs.

Authentication is used for multiple apps. After entering user credentials, select the apps you want authorized. The apps displayed in the list are imported apps with the Connect web service enabled.

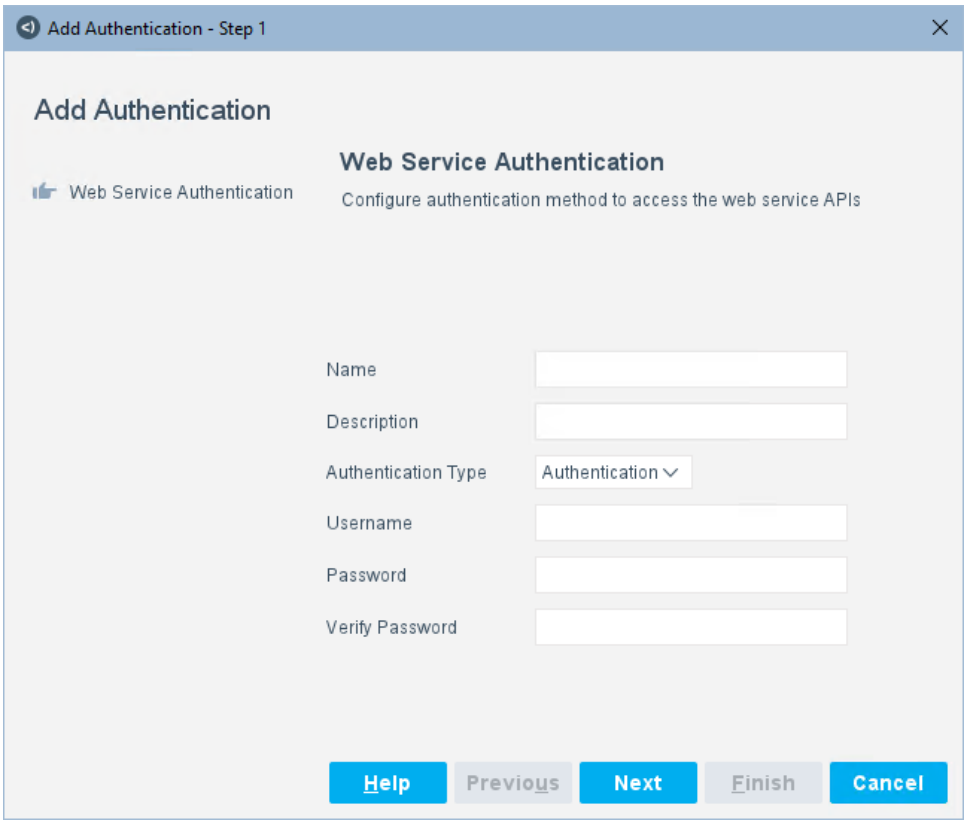
The user credentials configured for the apps will be used to obtain a JWT token through the Connect web service API. The JWT token will then be sent with subsequent Connect web service API requests. To learn more, see Use the Connect web service.

To configure web service authentication:

1. Select the Web Service Authentication tab in the Connect pane.



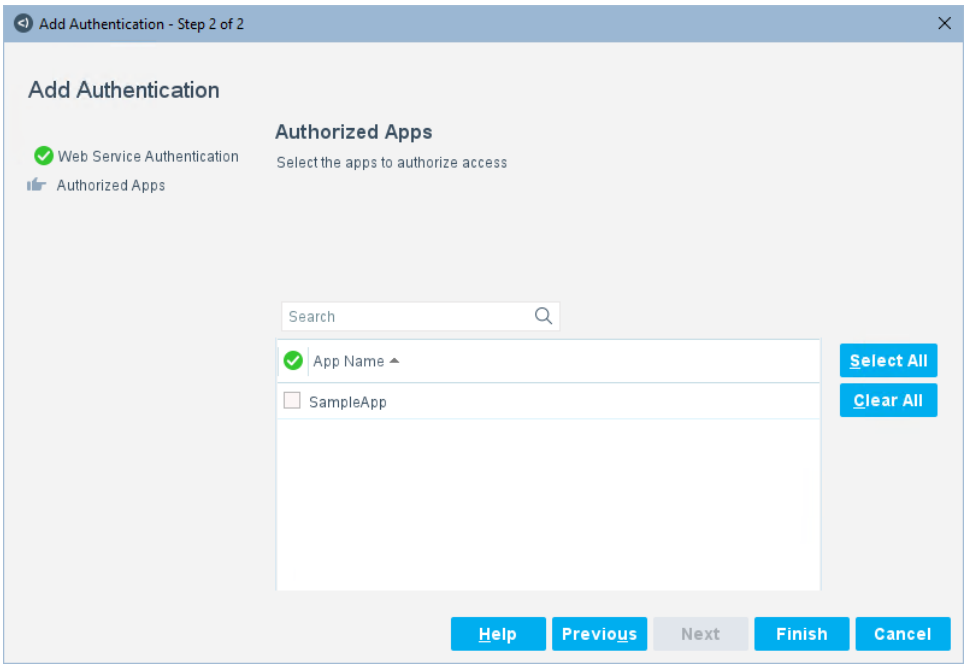
2. Select Add.



3. Configure the fields as follows:

Fields	Description
Name	(Required) Enter a unique name for the user.
Description	Enter a description for the user.
Authentication Type	Select the type of authentication. The valid type is Authentication.
Username	(Required) Enter a username.
Password	(Required) Enter a password.
Verify Password	Re-enter the password to verify it.

4. Select Next.

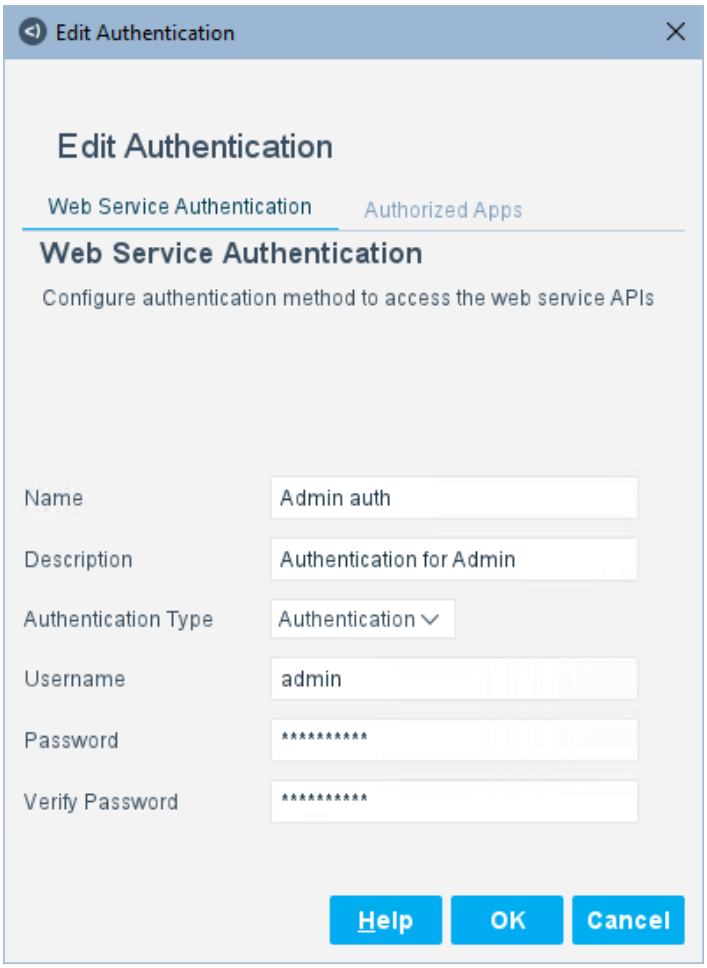


5. Select the checkbox for each app you want authorized or Select All to select all apps in the list.
6. Select Finish. The user is displayed in the Web Service Authentication tab.
7. Select Apply to save the configuration.

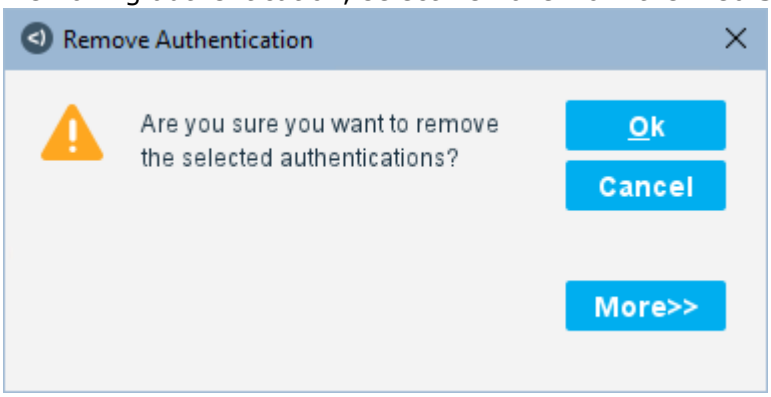
Edit or remove web service authentication

To edit:

1. Select an existing user in the Web Service Authentication tab and
2. Select Edit - there are tabs for each configuration pane:



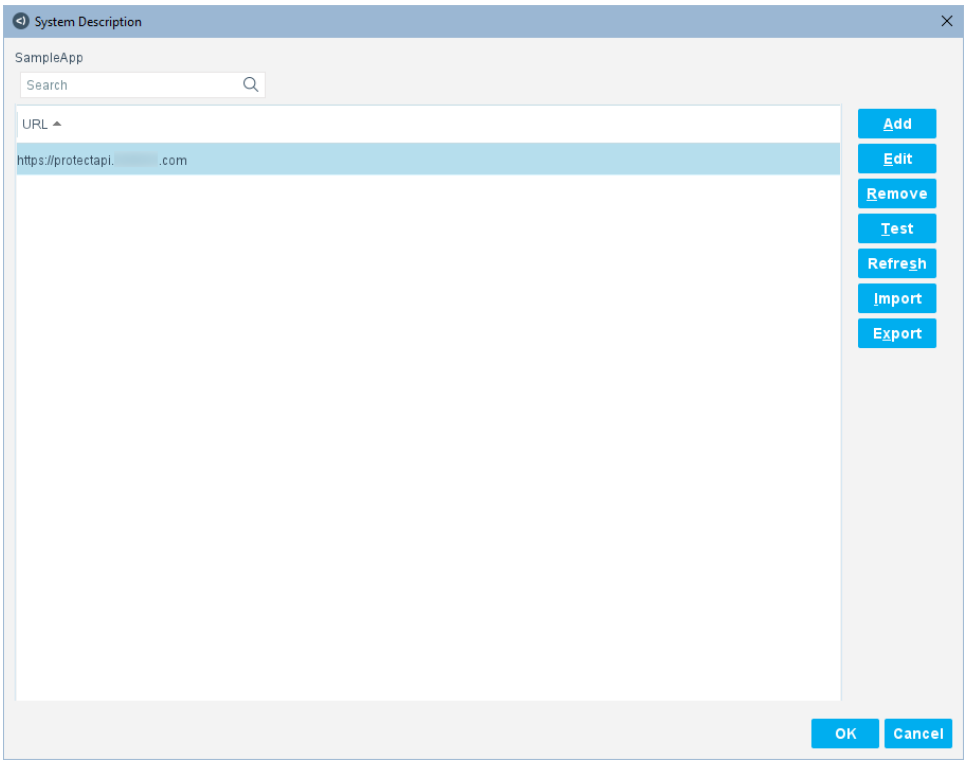
3. Select OK to close the dialog box
4. For removing authentication, select Remove from the Web Service Authentication tab:



5. Select More for details, or select OK

System description dialog box details

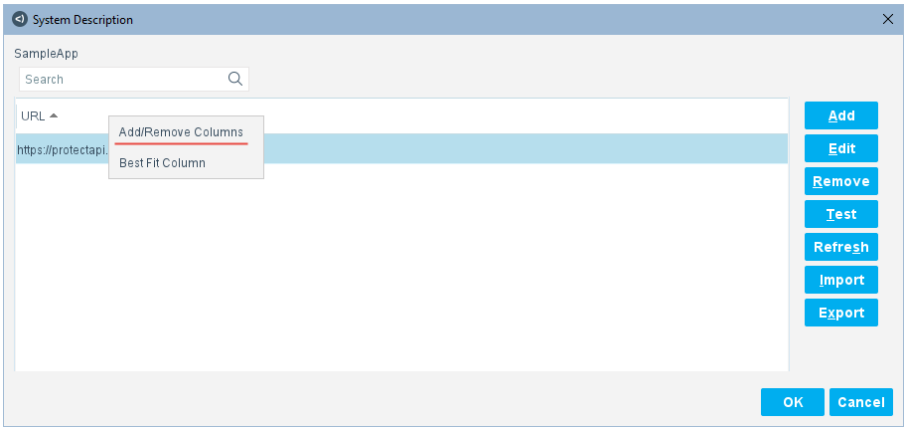
Once you've configured the system description, you'll see it displayed in the System Description dialog box:



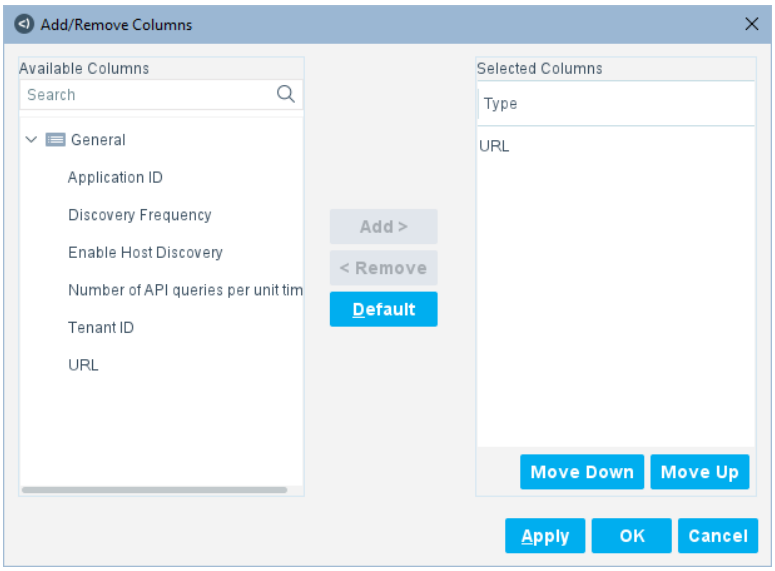
Columns in system description dialog box

This topic describes how users see columns in the System Description dialog box.

You can define columns through the system.conf file, by using the add to column parameter. When you right-click the column titles, a menu for adding and removing columns is displayed:



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



2. Move columns in the lists for Available Columns and Selected Columns, or use the Move Up and Move Down buttons to reorder the columns in the Add/Remove Columns dialog box
3. To delete a column, select it and select Remove Column in the System Description dialog box. You cannot remove columns if only one is present.
4. 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

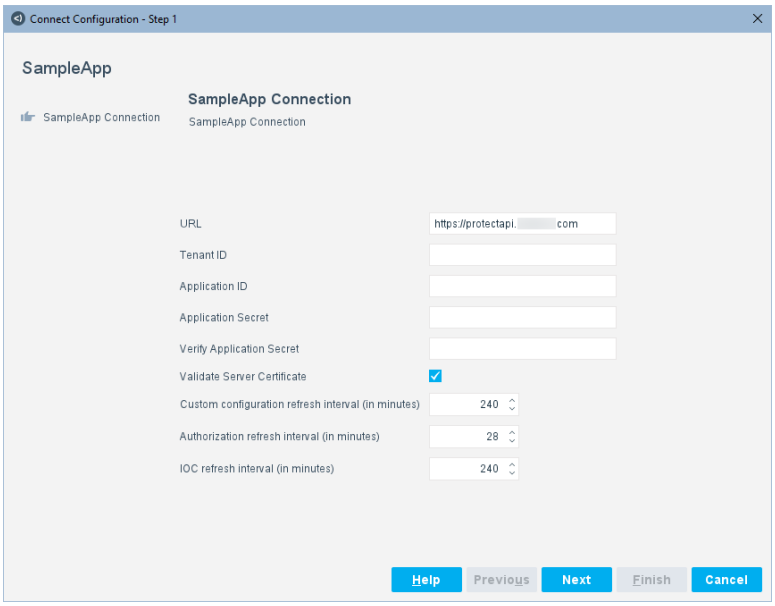
This topic describes how users see buttons in the System Description diaglog box.

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

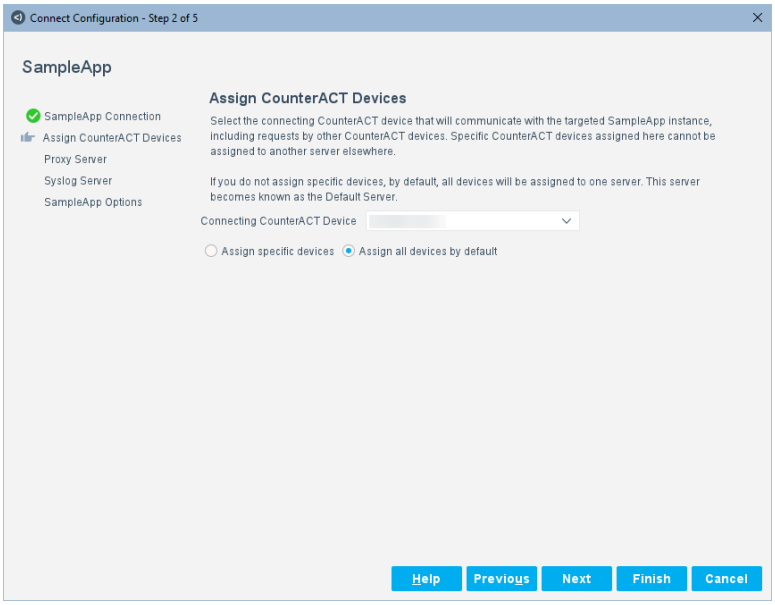
- Add
- Edit
- Remove
- Refresh
- plus OK and Cancel for changes made in the pane

Add a system description

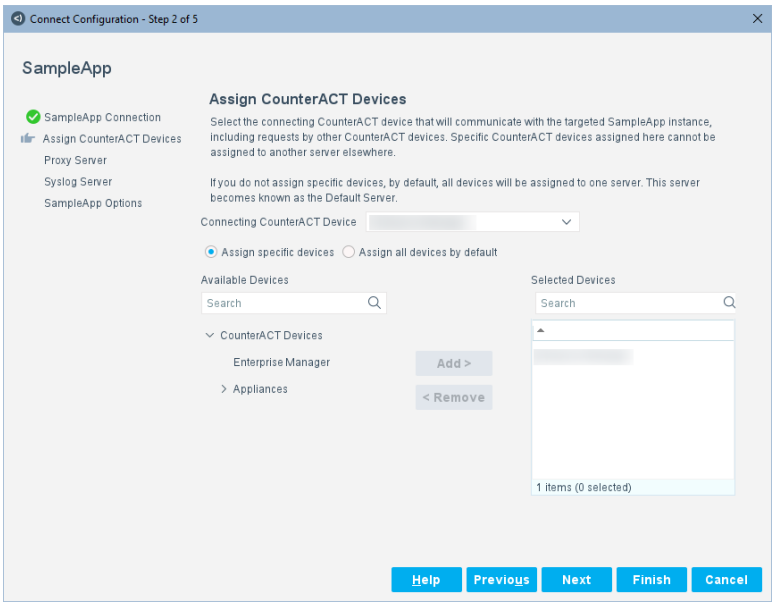
Select Add to display the first configuration panel. The number of panels in the system description are defined in the system.conf file:



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:



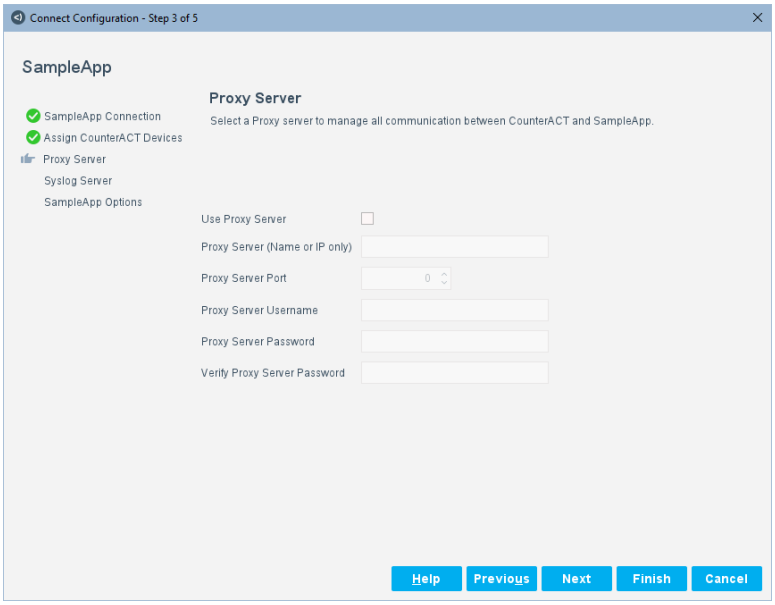
At first, the Assign CounterACT Devices panel has only one option, Assign all devices by default, and it is selected so that one device is added. If you want to add a second device, the Assign CounterACT Devices panel has more options:



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

- Connecting CounterACT Device via Enterprise Manager or an IP address: 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. It's not recommended to use the Enterprise Manager as the connecting CounterACT device. If necessary, make sure that it's 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, 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're 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.
- Additionally, it's important to note the following:
    - If you try to add a device that's already used, an error will display
    - 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.

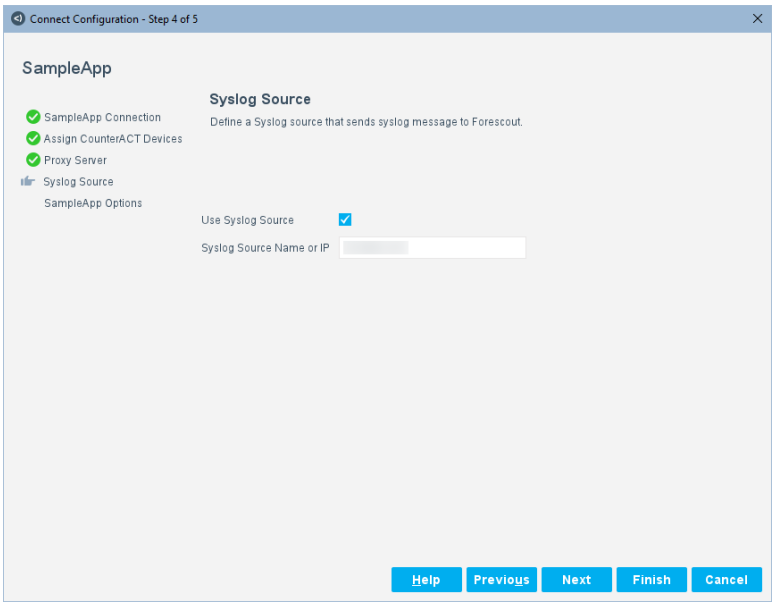
Select Next to display the next configuration panel that is defined in the system.conf file, such as, the predefined Proxy Server panel:



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 user name 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.

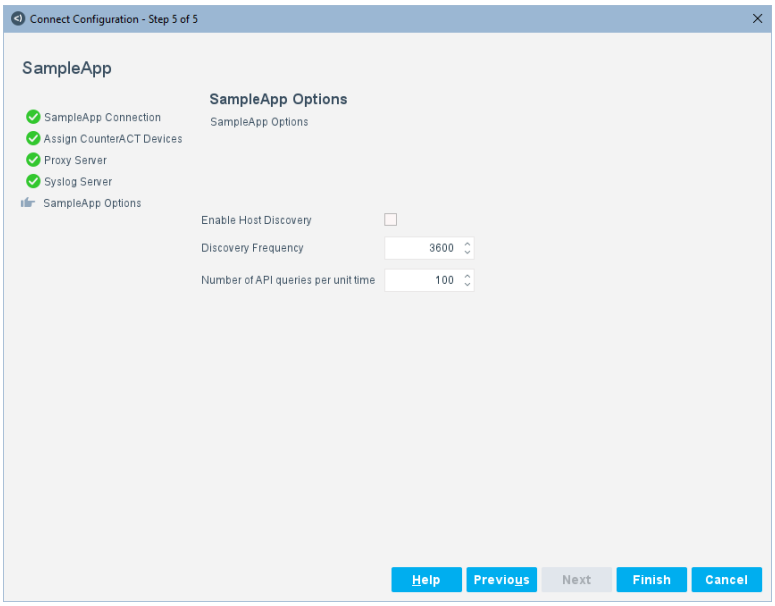
Select Next to display the next configuration panel that is defined in the system.conf file, such as the Syslog Source panel:



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

Use Syslog Source	Select the checkbox to enable or disable the syslog source. If Use Syslog Source is enabled, the Syslog Source Name or IP field is required. If Use Syslog Source is disabled, Connect does not get or process the syslog message from the specified syslog source.
Syslog Source Name or IP	Enter the name or IP address of the syslog source on which Connect listens for syslog messages.

Select Next to display the next configuration panel that is defined in the system.conf file, such as SampleApp Options:

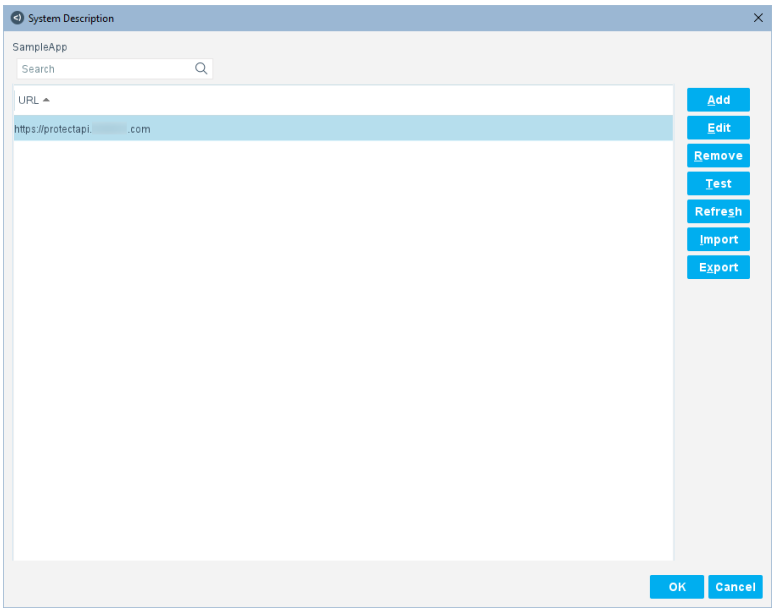


The user configuring the system description enters the following information for the predefined fields on the panel:

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, then select Apply to save the configuration.

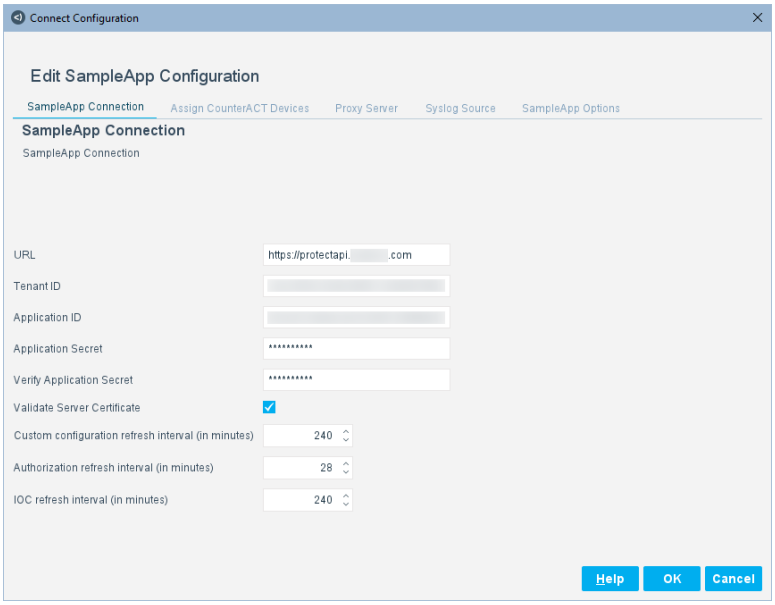
The configured system description is displayed in the System Description dialog box:



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

Edit a system description

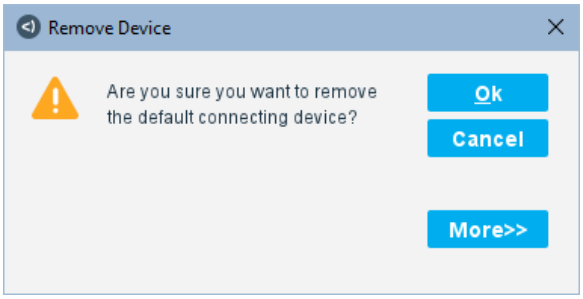
Select an existing system description in the System Description dialog box and select Edit. There are tabs for each panel:



Select Os to close the dialog box.

Remove a system description

Select an existing system description in the System Description dialog box and select Remove. A confirmation is displayed:



Select More for details or select OK.

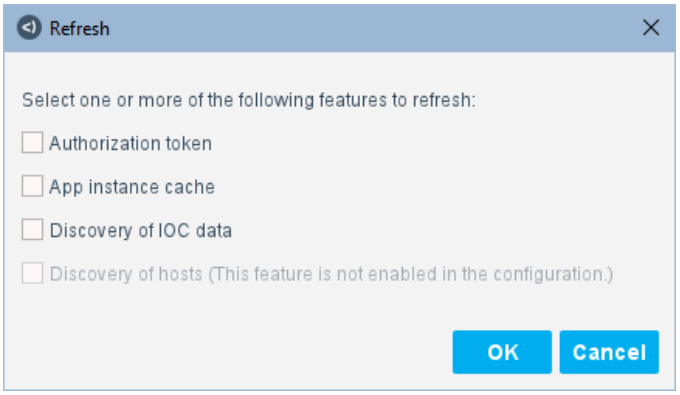
Scenarios for removing a system description are as follows:

Description	Result
The system description being removed has the connecting appliance as the default appliance.	If there is only one system description and it is the default, the remove is allowed.
The system description being removed has the connecting appliance as the default appliance.	If there are two system descriptions, the remove is allowed and the connecting appliance is assigned as the default appliance.

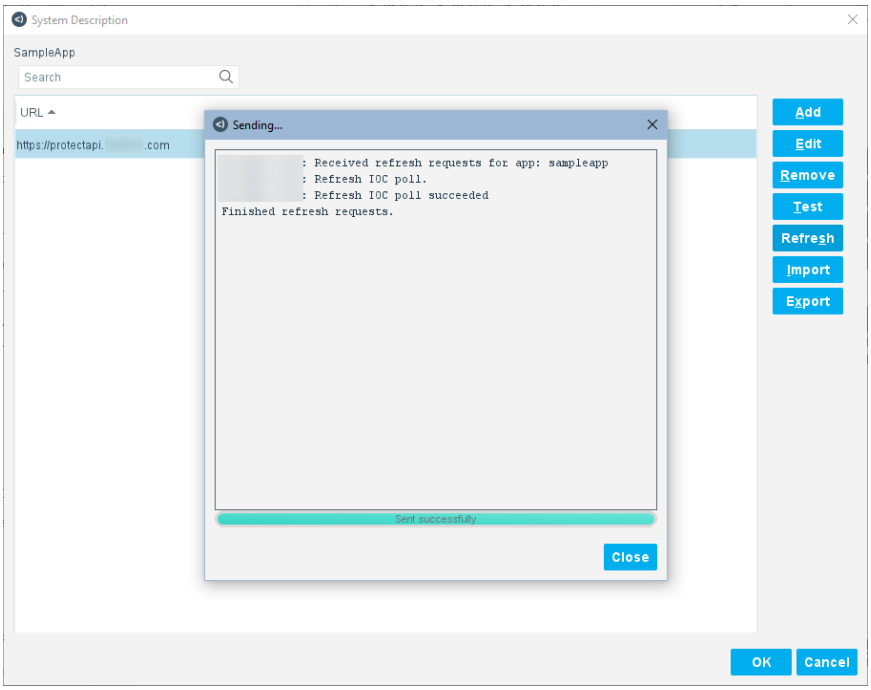
Description	Result
The system description being removed has the connecting appliance as the default appliance.	If there are more than two system descriptions, the remove is not allowed. You must select a new default before removing the system description.

Refresh app features

Select a system description in the System Description dialog box and select Refresh to refresh selected app features:



Select one or more app features to refresh and then select OK. For example, select Discovery of IOC data to manually refresh IOC data:



Depending on the app, up to four features can be selected to be refreshed:

- Authorization token
- App instance cache
- Discovery of IOC data
- Discovery of hosts

If a feature is not supported by the app, the Refresh dialog box will show the following text: (This feature is not supported in the app.)

If a feature is not enabled in the configuration of the app, the Refresh dialog box will show the following text: (This feature is not enabled in the configuration.)

If none of the features are supported by the app, the Refresh button is not enabled in the System Description dialog box.

If your app has all features and you select all of them to refresh, the refresh occurs in the order listed. Discovery of hosts is the last feature to refresh as it may take the most time. If you frequently refresh, the results of one refresh may not have completed before the next refresh is triggered.

Use the Refresh button to do a manual refresh separate from any schedule, such as a scheduled refresh of an authorization token or polling for host discovery. After a manual refresh, the timer is reset, so the next scheduled refresh will be postponed to the next refresh interval. For example, when polling is scheduled for every four hours and the next scheduled discovery is expected to be at 8:00 AM, if there is a manual refresh two hours before at 6:00 AM, the next poll will occur four hours later, at 10:00 AM.

Note:
A manual refresh of app instance cache does not impact its schedule.

To enable Authorization token, you must have the following defined in the system.conf file for the app:

```
"authorization":true,
```

To enable App instance cache, you must have the following defined in the system.conf file for the app:

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

To enable Discovery of IOC data, you must have the following defined in the system.conf file for the app:

```
"ioc_poll":true,
```

To enable Discovery of hosts, you must have the following defined in the system.conf file for the app:

```
"host discovery":true,
```

In addition, for Discovery of hosts, the Enable Host Discovery checkbox must be selected when the system description for the app is configured.

For details, see the following sections:

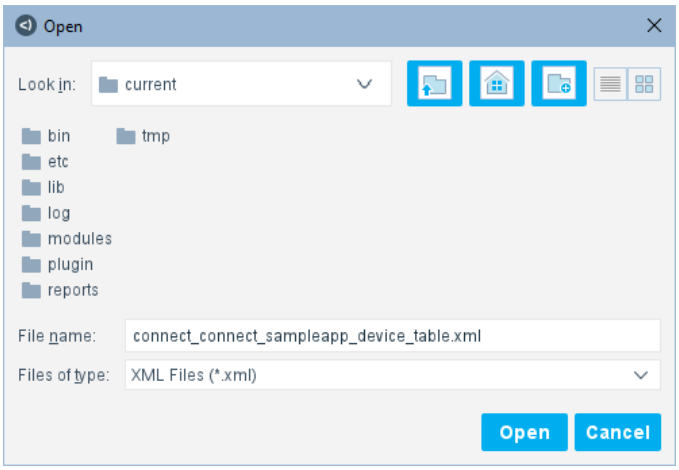
- “authorization” Field
- “app_instance_cache” Field
- “ioc_poll” Field
- “host discovery” Field

Import a system description

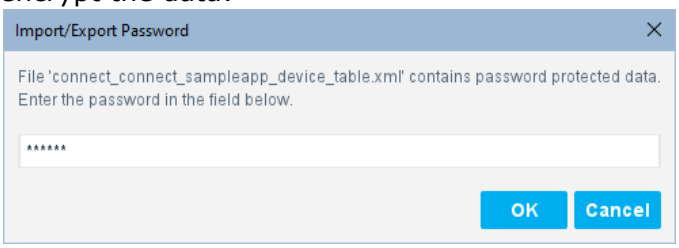
To import a system description:

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

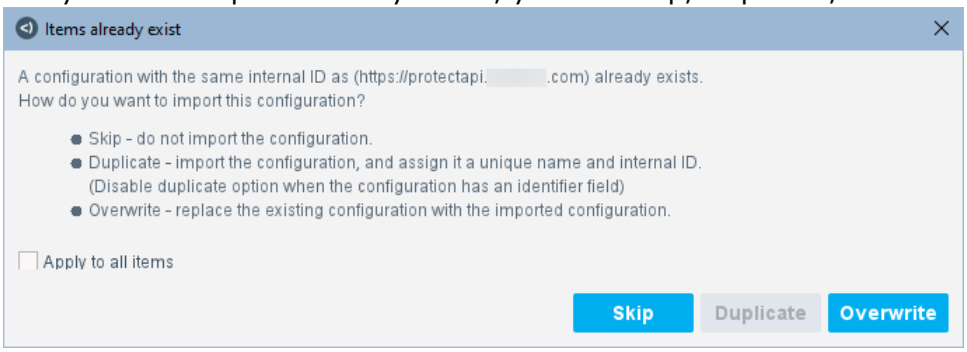
2. To change the folder location, select Browse:



3. Select Open, then 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:

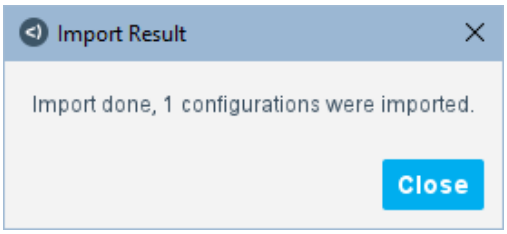


   b. Enter the password that you used when you exported the configuration and then select OK
4. If the system description already exists, you can Skip, Duplicate, or Overwrite it:



   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.

5. The result of the import is displayed, then you can select Close:



Scenarios for import

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

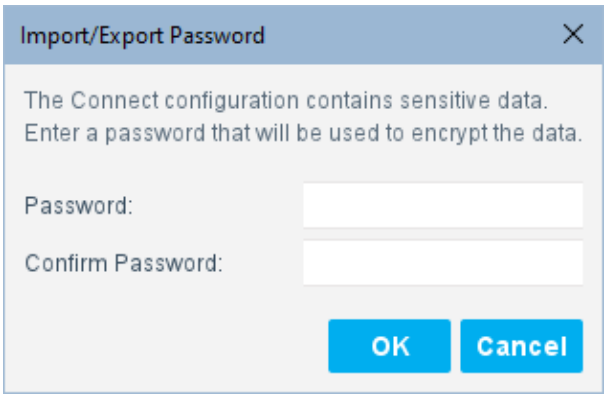
Description	Result
The imported system description is the first configuration of this app.	The import is allowed and the imported focal appliance is set as the default.
The imported system description will overwrite the default appliance because the user selected Overwrite during the import.	The import is allowed and the imported focal appliance is set to the default.
The imported system description has the default appliance assigned, the default is correct, and is not a duplicate appliance.	If no system description has been configured yet, the import is allowed.
The imported system description has the default appliance assigned, the default is correct, and is not a duplicate appliance.	If more than one system description is configured, the import is allowed and the imported focal appliance is switched to a specific appliance.
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.	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.
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.	If more than one system description is configured and if all other appliances have been assigned to other devices, the import is not allowed.
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.	If more than one system description is configured and if an appliance is available, one is selected at random.

Description	Result
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.	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.
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.	If all the specific appliances are not found and an appliance is available, one is selected at random.
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.	If some of the specific appliances are found, the import of the correct specific appliances is allowed.
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.	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

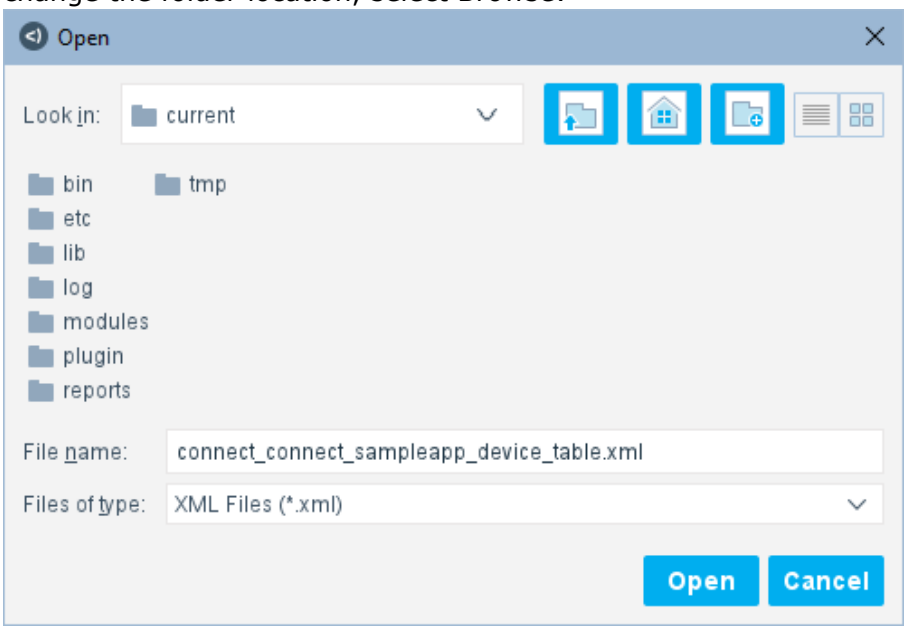
To export a system description:

1. Select Export to save a backup of the configuration. The only supported format is .xml.
   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:

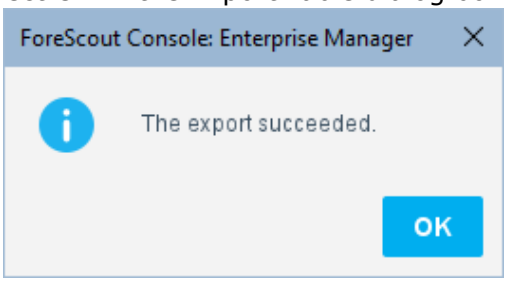


   b. Enter the password
2. The Export Table dialog box opens. You can select to export the Selected rows only:

3. To change the folder location, select Browse:



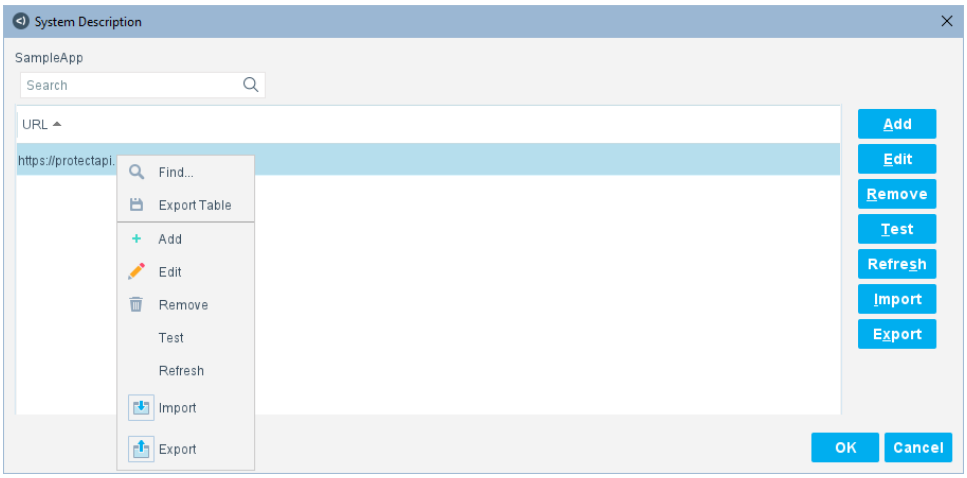
4. Select Open
5. Select OK in the Export Table dialog box. Lastly, select Close:



Menu in system description dialog box

This topic describes the menu in the System Description dialog box.

You can access a menu by right-clicking an existing system description in the System Description dialog box:

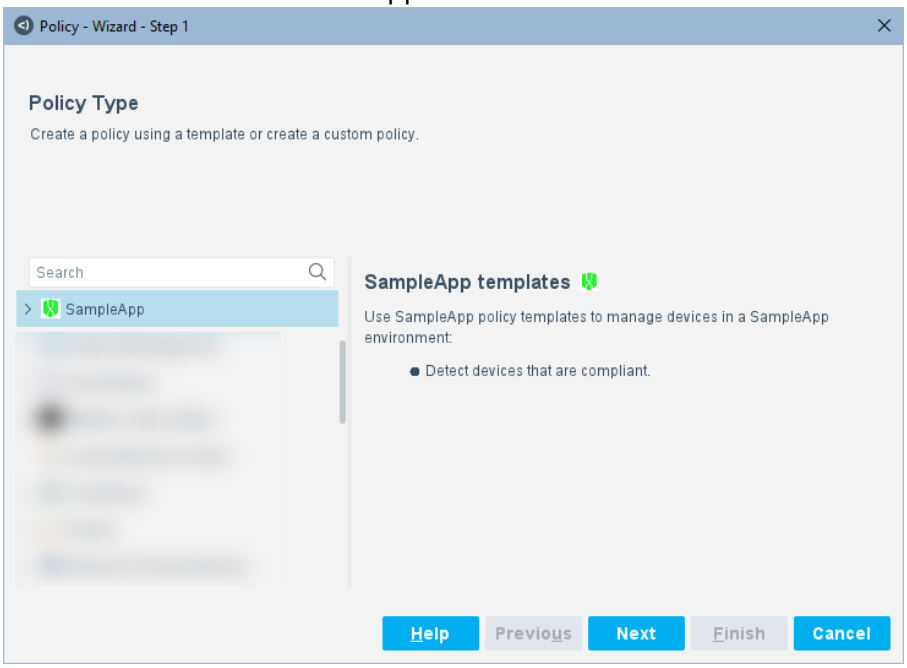


Configure policy templates in Connect

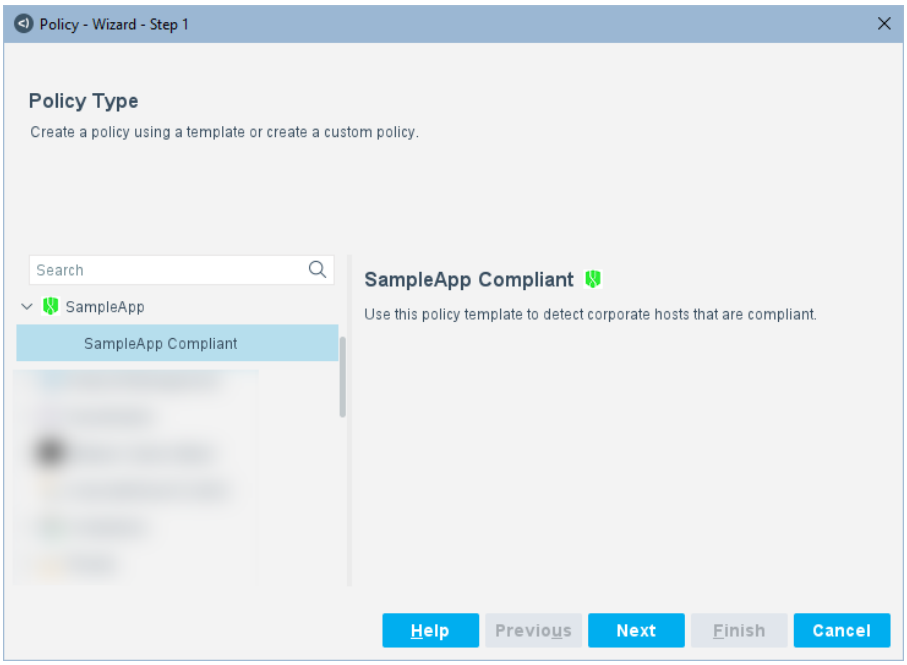
This topic describes configuring policy templates in Connect.

To configure a policy template:

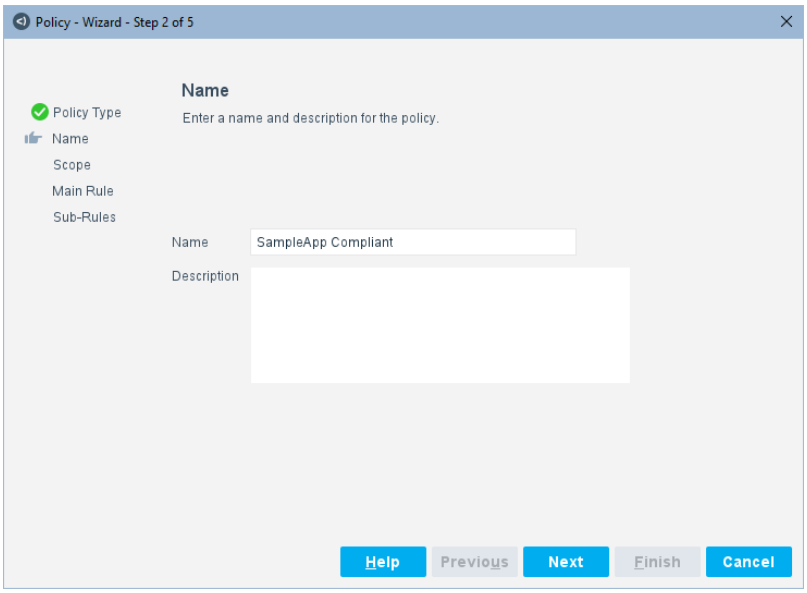
1. In the Forescout Console, select Policy
2. Select Add and search for the app name:



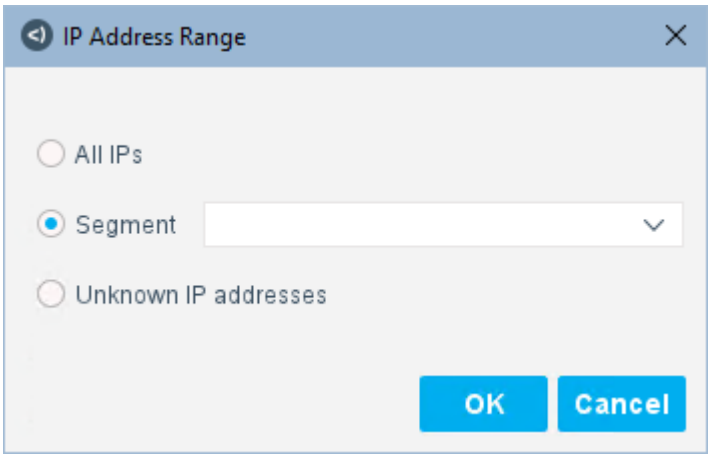
3. Expand the folder and select a policy template:



4. Select Next:



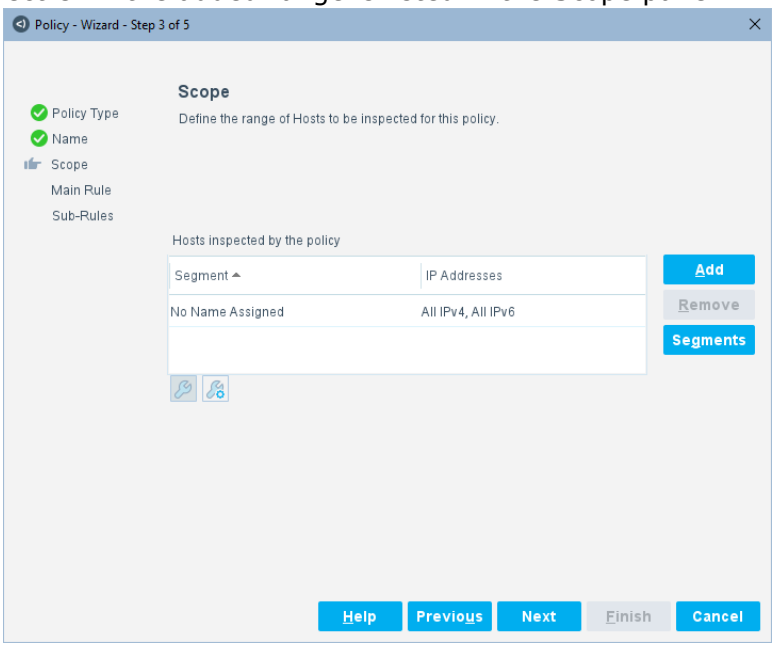
5. Enter a name for the policy and select Next. Both the IP Address Range dialog box and the Scope pane opens.
6. Use the dialog box to define which endpoints are inspected:



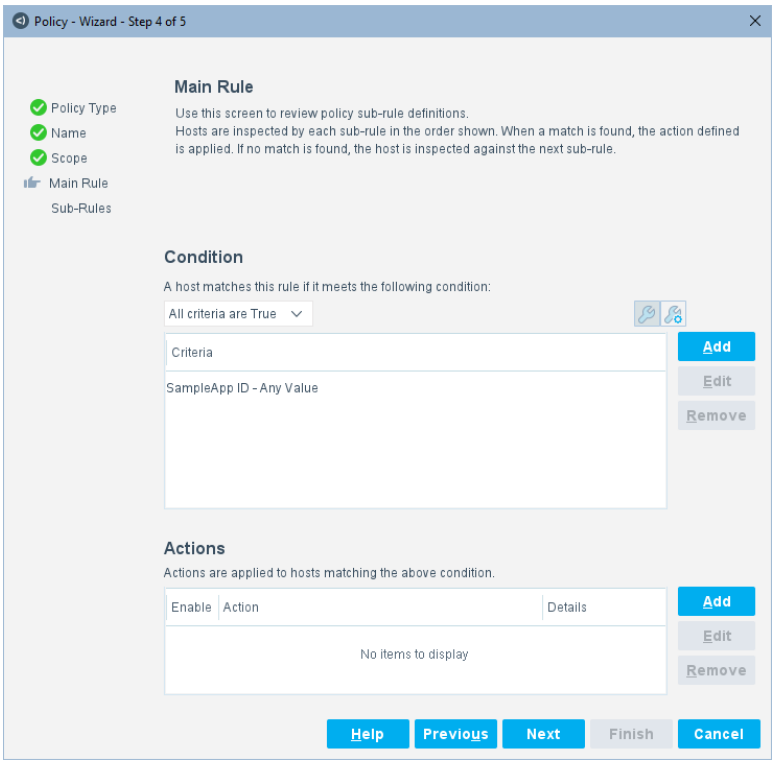
The following options are available:

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.

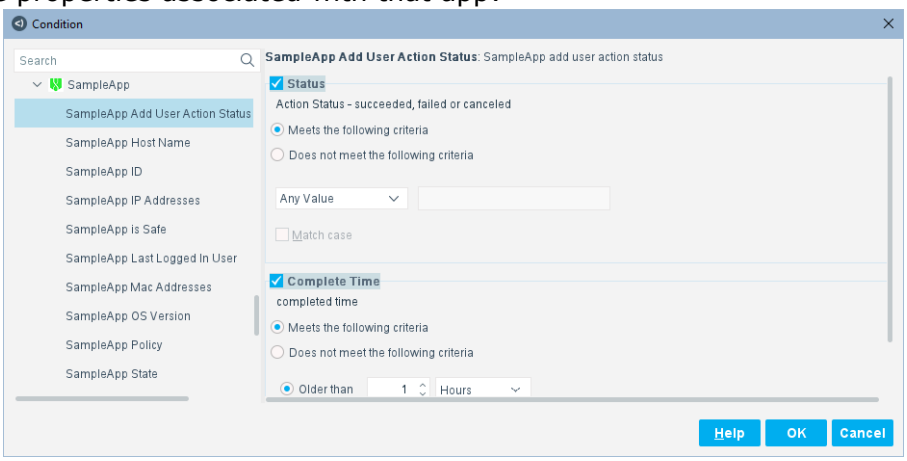
7. Select OK - the added range is listed in the Scope pane:



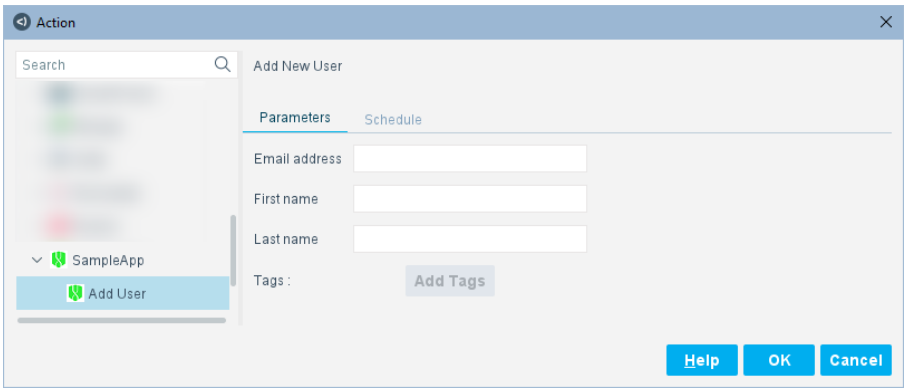
8. Select Next:



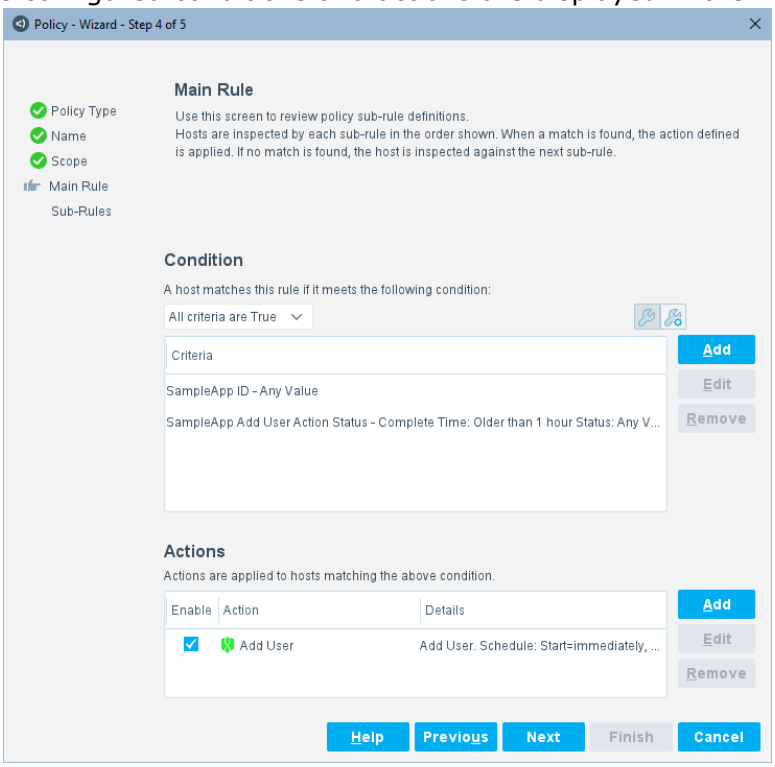
9. To add a condition, select Add in the Condition area and search for the app name to see the properties associated with that app:



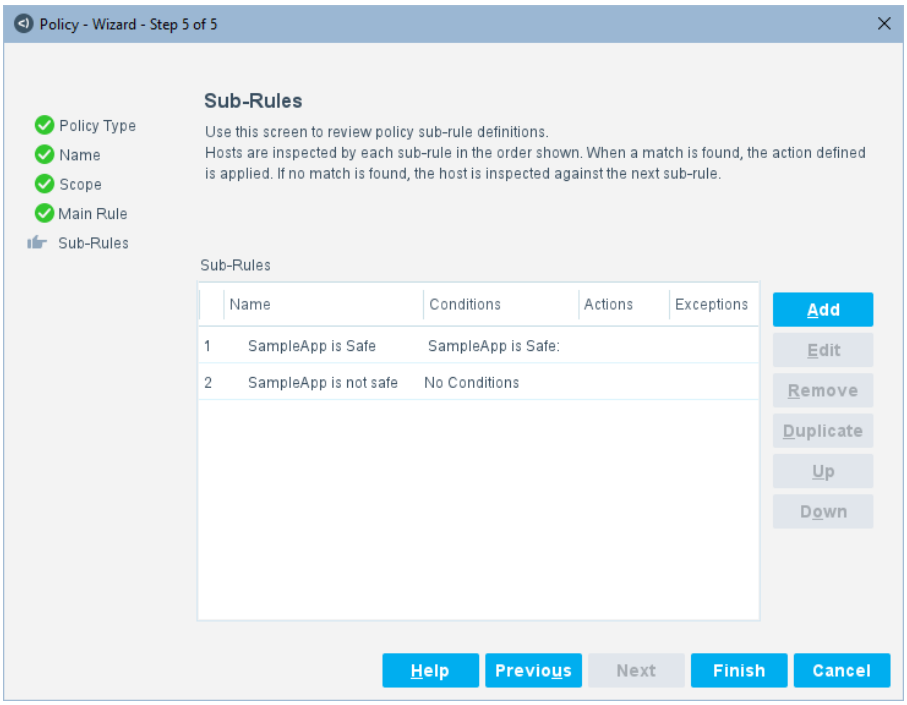
10. Configure conditions for the policy and then select OK
11. To add an action, select Add in the Actions area and search for the app name to see the actions associated with that app:



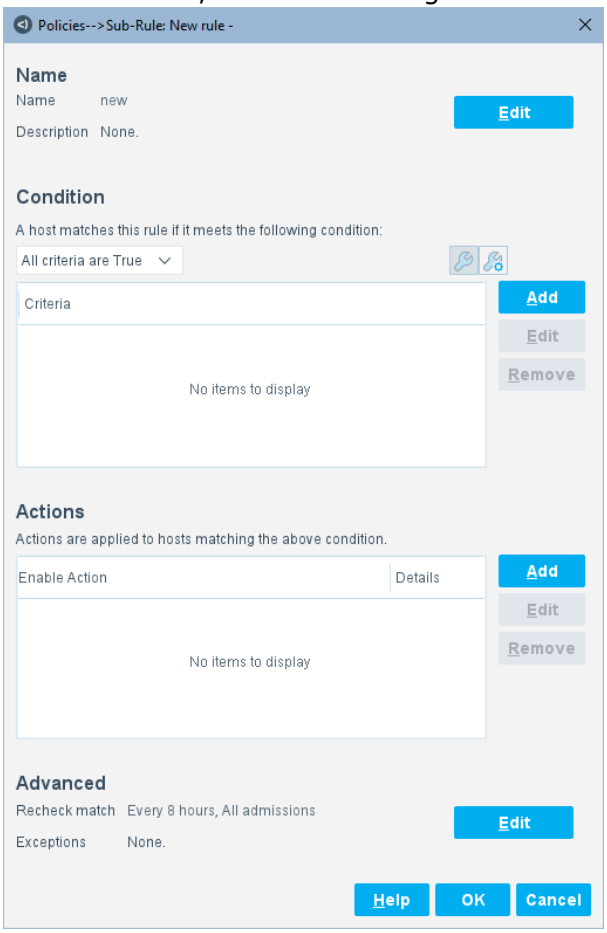
12. Configure actions for the policy and then select OK
13. The configured conditions and actions are displayed in the Main Rule:



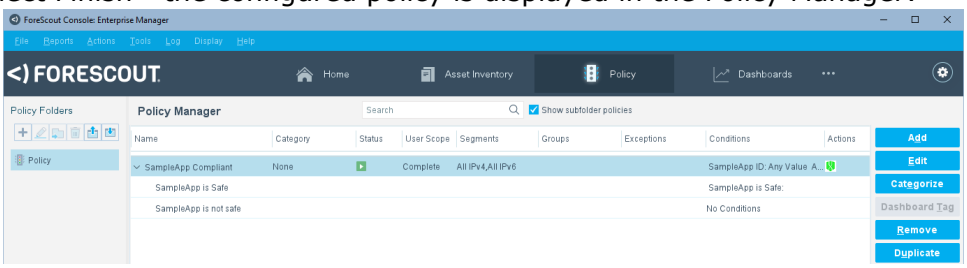
14. To configure sub-rules, select Next:



15. To add a sub-rule, select Add and give the sub-rule a name:



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

This topic provides a sample system.conf file.

```
{
   "name":"SampleApp",
   "version":"1.1.1",
   "author":"Concert Masters",
   "testEnable":true,
   "web service":true,
   "focal only syslog": true,
   "panels":[
      {
         "title":"SampleApp Connection",
         "description":"SampleApp Connection",
         "fields":[
            {
               "display":"URL",
               "field ID":"connect_sampleapp_url",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"true",
               "identifier":"true",
               "tooltip":"URL",
               "value":"https://protectapi.cylance.com"
```

```
            },
            {
               "display":"Tenant ID",
               "field ID":"connect_sampleapp_tenant_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Tenant ID",
               "value":"bfd64dee-5834-43d4-806f-1c0d28975fbc"
            },
            {
               "display":"Application ID",
               "field ID":"connect_sampleapp_application_id",
               "type":"shortString",
               "mandatory":"true",
               "add to column":"true",
               "show column":"false",
               "tooltip":"Application ID",
                           "value":"6a116084-6a79-4ba0-ad7d-595125888b35"
            },
            {
               "display":"Application Secret",
               "field ID":"connect_sampleapp_application_secret",
               "type":"encrypted",
               "mandatory":"true",
               "tooltip":"Application Secret",
               "value":"17f07b2b-b1ae-458e-be37-5dfb49ef08bb"
            },
            {
               "certification validation":true
            },
                     {
                  "app_instance_cache":true,
                      "display":"Custom configuration refresh interval
(in minutes)",
                  "min":5,
                  "max":2400,
                      "value":240
               },
                      {
                  "authorization":true,
                      "display":"Authorization refresh interval (in
minutes)",
```

```
                    "min":1,
                    "max":100,
                        "value":28
                },
            {
               "ioc_poll":true,
               "display":"IOC refresh interval (in minutes)",
               "min":5,
               "max":2400,
               "value":240
            }
                 ]
      },
      {
         "focal appliance":true,
         "title":"Assign CounterACT Devices",
         "description":"<html>Select the connecting CounterACT device that
will communicate with the targeted SampleApp instance, including requests
by other CounterACT devices. Specific CounterACT devices assigned here
cannot be assigned to another server elsewhere.<br><br>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.<html>"
      },
      {
         "proxy server":true,
         "title":"Proxy Server",
         "description":"<html>Select a Proxy server to manage all
communication between CounterACT and SampleApp.↙html>"
      },
      {
         "syslog source":true,
         "title":"Syslog Source",
         "description":"<html>Define a Syslog source that sends syslog
message to Forescout.↙html>"
      },
      {
         "title":"SampleApp Options",
         "description":"SampleApp Options",
         "fields": [
           {
               "host discovery": true,
               "display":"Discovery Frequency",
               "max":300000,
               "add to column":"true",
```

```
                "show column":"false",
                "value":3600
            },
            {
                "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
            }
        ]
    }
  ]
}
```

Sample property.conf file

This topic provides a sample property.conf file.

```
{
  "name": "SampleApp",
  "groups": [
    {
      "name":"connect_sampleapp_sampleapp",
      "label":"SampleApp"
    }
  ],
  "properties": [
    {
      "tag": "connect_sampleapp_state",
      "label": "SampleApp State",
      "description": "SampleApp State",
      "type": "string",
      "web_enable": true,
      "options": [
        {
          "name": "Online",
          "label": "Online"
        },
        {
          "name": "Offline",
```

```
        "label": "Offline"
      }
    ],
    "group": "connect_sampleapp_sampleapp",
    "resolvable": true,
    "require_host_access": false,
    "inventory": {
      "enable": true,
      "description": "Inventory of SampleApp State"
    },
    "asset_portal": true,
    "track_change": {
      "enable": true,
      "label": "SampleApp State Changed",
      "description": "Track Change property for SampleApp state"
    },
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_last_logged_in_user",
    "label": "SampleApp Last Logged In User",
    "description": "SampleApp Last Logged In User",
    "type": "string",
    "web_enable": true,
    "group": "connect_sampleapp_sampleapp",
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_mac_addresses",
    "label": "SampleApp Mac Addresses",
    "description": "SampleApp Mac Addresses",
    "type": "string",
```

```
    "group": "connect_sampleapp_sampleapp",
    "list": true,
    "web_enable": false,
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_ip_addresses",
    "label": "SampleApp IP Addresses",
    "description": "SampleApp IP Addresses",
    "type": "string",
    "group": "connect_sampleapp_sampleapp",
    "list": true,
    "overwrite": true,
    "web_enable": false,
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_is_safe",
    "label": "SampleApp is Safe",
    "description": "SampleApp is Safe",
    "type": "boolean",
    "group": "connect_sampleapp_sampleapp",
    "web_enable": true,
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
```

```
    "tag": "connect_sampleapp_host_name",
    "label": "SampleApp Host Name",
    "description": "SampleApp Host Name",
    "type": "string",
    "group": "connect_sampleapp_sampleapp",
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_os_version",
    "label": "SampleApp OS Version",
    "description": "SampleApp OS Version",
    "type": "string",
    "group": "connect_sampleapp_sampleapp",
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_id",
    "label": "SampleApp ID",
    "description": "SampleApp ID",
    "type": "string",
    "group": "connect_sampleapp_sampleapp",
    "dependencies": [
      {
        "name": "mac",
        "redo_new": true,
        "redo_change": true
      }
    ]
  },
  {
    "tag": "connect_sampleapp_policy",
    "label": "SampleApp Policy",
```

```
    "description": "SampleApp Policy",
    "type": "composite",
    "group": "connect_sampleapp_sampleapp",
    "web_enable": true,
    "inventory": {
      "enable": true,
      "description": "Inventory of SampleApp 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
      }
    ]
  },
  {
    "tag": "connect_sampleapp_add_user_action",
    "label": "SampleApp Add User Action Status",
    "description": "SampleApp add user action status",
    "type": "composite",
    "resolvable": false,
    "group": "connect_sampleapp_sampleapp",
    "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
      }
    ]
  },
  {
    "tag": "connect_sampleapp_threat_info",
    "label": "SampleApp Threat Information",
    "description": "SampleApp Threat Information",
    "type": "composite",
    "group": "connect_sampleapp_sampleapp",
    "list": true,
    "overwrite": true,
    "resolvable": false,
    "subfields": [
      {
        "tag": "name",
        "label": "Threat Name",
        "description": "Threat Name",
        "type": "string"
      },
      {
        "tag": "date",
        "label": "Reported Date",
        "description": "Reported Date",
        "type": "date"
      },
      {
        "tag": "file_name",
        "label": "File Name",
        "description": "File Name",
        "type": "string"
```

```
        },
        {
          "tag": "hash",
          "label": "Hash",
          "description": "Hash",
          "type": "string"
        },
        {
          "tag": "hash_type",
          "label": "Hash Type",
          "description": "Hash Type",
          "type": "string"
        },
        {
          "tag": "severity",
          "label": "Severity",
          "description": "Severity",
          "type": "string"
        }
      ]
    }
  ],
  "action_groups": [
    {
      "name":"connect_sampleapp_sampleapp",
      "label":"SampleApp"
    }
  ],
  "actions": [
    {
      "name": "connect_sampleapp_add_user",
      "label": "Add User",
      "group": "connect_sampleapp_sampleapp",
      "description": "Add New User",
      "ip_required": false,
      "threshold_percentage": 1,
      "params": [
        {
          "name": "sampleapp_email",
          "label": "Email address",
          "description": "SampleApp email address",
          "type": "string"
        },
        {
```

```
            "name": "sampleapp_first_name",
            "label": "First name",
            "description": "SampleApp first name",
            "type": "string"
          },
          {
            "name": "sampleapp_last_name",
            "label": "Last name",
            "description": "SampleApp last name",
            "type": "string"
          }
        ],
        "dependencies": [
          {
            "name": "mac",
            "redo_new": true,
            "redo_change": true
          }
        ],
        "undo": {
          "label": "Cancel SampleApp Add User",
          "description": "Remove Added User"
        }
      }
    ],
    "scripts": [
      {
        "name": "sampleapp_resolve.py",
        "properties": [
          "connect_sampleapp_state",
          "connect_sampleapp_last_logged_in_user",
          "connect_sampleapp_mac_addresses",
          "connect_sampleapp_is_safe",
          "connect_sampleapp_id"
        ]
      },
      {
        "name": "sampleapp_ioc_resolve.py",
        "properties": [
          "connect_sampleapp_host_name",
          "connect_sampleapp_os_version"
        ]
      },
      {
```

```
      "name": "sampleapp_add_user.py",
      "actions": [
        "connect_sampleapp_add_user"
      ]
    },
    {
      "name": "sampleapp_delete_user.py",
      "is_cancel": true,
      "actions": [
        "connect_sampleapp_add_user"
      ]
    },
    {
      "name": "sampleapp_test.py",
      "test": true
    },
    {
      "name": "sampleapp_poll.py",
      "discovery": true
    },
        {
          "name":"sampleapp_authorization.py",
          "authorization":true
        },
        {
          "name":"sampleapp_parse_syslog_message.py",
          "syslog_message":true
    },
    {
      "name":"sampleapp_users.py",
      "app_instance_cache": true
    },
    {
      "name":"sampleapp_ioc_poll.py",
      "ioc_poll": true
    }
  ],
  "policy_template": {
    "policy_template_group": {
       "name": "connect_sampleapp",
       "label": "SampleApp",
       "display": "SampleApp",
       "description": "SampleApp templates",
       "full_description": "<html>Use SampleApp policy templates to manage
```

```
devices in a SampleApp environment:<ul><li>Detect devices that are
compliant.↙li>↙ul>↙html>",
       "title_image": "connect_sampleapp.png"
    },
    "policies": [
      {
        "name": "connect_sampleapp_compliant",
        "label": "SampleApp Compliant",
        "display": "SampleApp Compliant",
        "help": "SampleApp Compliant Policy",
        "description": "Creates SampleApp compliant policies",
        "file_name": "SampleAppCompliance.xml",
        "full_description": "<html>Use this policy template to detect
corporate hosts that are compliant.↙html>",
        "title_image": "connect_sampleapp.png"
      }
    ]
  }
}
```

Sample policy template .xml file for Connect

This topic provides a sample policy template .xml file for Connect.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<RULES>
    <RULE APP_VERSION="8.2.2-731" CACHE_TTL="259200"
CACHE_TTL_SYNCED="true" CLASSIFICATION="REG_STATUS" DESCRIPTION=""
ENABLED="true" ID="822642380502786913" NAME="SampleApp Compliant"
NOT_COND_UPDATE="true" UPGRADE_PERFORMED="true">
        <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: SampleApp Compliant-→
            <CONDITION EMPTY_LIST_VALUE="false"
FIELD_NAME="connect_sampleapp_id" LABEL="SampleApp ID"
LEFT_PARENTHESIS="0" LOGIC="AND" PLUGIN_NAME="Connect"
PLUGIN_UNIQUE_NAME="connect_module" PLUGIN_VESRION="1.6.0"
PLUGIN_VESRION_NUMBER="16000165" RET_VALUE_ON_UKNOWN="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>
        <RANGE FROM="0.0.0.0" TO="255.255.255.255"↗
        <SUBNET address="::" prefix="0"↗
        <RULE_CHAIN>
            <INNER_RULE APP_VERSION="8.2.2-731" CACHE_TTL="259200"
CACHE_TTL_SYNCED="true" CLASSIFICATION="REG_STATUS" DESCRIPTION=""
ID="3623802004997893115" NAME="SampleApp is Safe" NOT_COND_UPDATE="true"
RECHECK_MAIN_RULE_DEF="true">
                <MATCH_TIMING RATE="28800" SKIP_INACTIVE="true">
                    <ADMISSION ALL="true"↗
                ↙MATCH_TIMING>
                <EXPRESSION EXPR_TYPE="SIMPLE">
                    <!--Rule expression. Rule name is: SampleApp is Safe-→
                    <CONDITION EMPTY_LIST_VALUE="false"
FIELD_NAME="connect_sampleapp_is_safe" LABEL="SampleApp is Safe"
LEFT_PARENTHESIS="0" LOGIC="AND" PLUGIN_NAME="Connect"
PLUGIN_UNIQUE_NAME="connect_module" PLUGIN_VESRION="1.6.0"
PLUGIN_VESRION_NUMBER="16000165" RET_VALUE_ON_UKNOWN="UNMATCH"
RIGHT_PARENTHESIS="0">
                        <FILTER FILTER_ID="-7261535076996746710"
VALUE="true"↗
                    ↙CONDITION>
                ↙EXPRESSION>
            ↙INNER_RULE>
            <INNER_RULE APP_VERSION="8.2.2-731" CACHE_TTL="259200"
CACHE_TTL_SYNCED="true" CLASSIFICATION="REG_STATUS" DESCRIPTION=""
ID="7945390469234880573" NAME="SampleApp is not safe"
NOT_COND_UPDATE="true" RECHECK_MAIN_RULE_DEF="true">
                <MATCH_TIMING RATE="28800" SKIP_INACTIVE="true">
                    <ADMISSION ALL="true"↗
```

```
                ↙MATCH_TIMING>
            ↙INNER_RULE>
        ↙RULE_CHAIN>
        <REPORT_TABLES↗
    ↙RULE>
↙RULES>
```

Sample Connect script files

This topic provides sample Connect script files.

The following sample Python scripts, which are included in the sample app, assume the Proxy Server panel is enabled in the system.conf file. The scripts use the Connect Proxy Server module to handle all API calls.

If you do not have a Proxy Server requirement, use standard libraries to make API calls.

For Proxy Server panel details, see Proxy Server Panel Details. For Connect Proxy Server module details, see Appendix B: Connect Proxy Server Module.

The following sample Python scripts 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 IOC Poll Script for Connect
- Sample IOC Resolve Script for Connect
- Sample Parse Syslog Message Script for Connect

Sample test script for Connect

This topic provides a sample test script.

```
# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
jwt_token = params.get("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 jwt_token:
```

```
    response["succeeded"] = True
    response["result_msg"] = "Successfully connected."
else:
    response["succeeded"] = False
    response["result_msg"] = "Could not connect to SampleApp server."
```

Sample polling script for Connect

This topic provides a sample polling script.

The following sample script uses urlib.request in the Connect Proxy Server module to invoke the polling request.

You can also look at other sample scripts that use requests session in the Connect Proxy Server module to invoke the call.

```
import urllib.request
from connectproxyserver import ConnectProxyServer, ProxyProtocol

# Mapping between SampleApp API response fields to CounterACT properties
sampleapp_to_ct_props_map = {
    "state": "connect_sampleapp_state",
    "mac_addresses": "connect_sampleapp_mac_addresses",
    "id": "connect_sampleapp_id"
}

# CONFIGURATION
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
endpoints = []
logging.debug("SampleApp polling.")
# Check if we have valid auth token or not before processing.
if params.get("connect_authorization_token"):
    # ***** PART 2 - QUERY FOR DEVICES  ***** #
    jwt_token = params.get("connect_authorization_token")
    get_mac_url = url + "/devices/v2/"
    device_headers = {"Content-Type": "application/json; charset=utf-8",
                      "Authorization": "Bearer " + str(jwt_token)}

    try:
        # Create proxy server
        proxy_server = ConnectProxyServer(params)
        # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_context
```

```
        opener =
proxy_server.get_urllib_request_https_opener(ProxyProtocol.all,
                                                                    ssl_context)
        poll_request = urllib.request.Request(get_mac_url,
                                              headers=device_headers)
        # Use opener.open to get the request ( you can use urlopen as
well, look at resolve script
        poll_response = opener.open(poll_request)
        poll_response_json =
json.loads(poll_response.read().decode("utf-8"))
        logging.debug(
            "Response json: {}".format(json.dumps(poll_response_json)))

        # 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
        # {"property_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 poll_response_json.get("page_items"):
            endpoint = {}
            mac_with_dash = endpoint_data.get("mac_addresses")[0]
            mac = "".join(mac_with_dash.split("-"))
            endpoint["mac"] = mac
            properties = {}
            for key, value in endpoint_data.items():
                if key in sampleapp_to_ct_props_map and key is not
"mac_addresses":
                    properties[sampleapp_to_ct_props_map[key]] = value
            endpoint["properties"] = properties
            endpoints.append(endpoint)
        response["endpoints"] = endpoints
    except Exception as e:
        response["error"] = "Could not retrieve endpoints."
```

```
        logging.debug("Get error: {}".format(str(e)))
else:
    response["error"] = "Unauthorized"
```

Sample resolve script for Connect

This topic provides a sample resolve script.

The following sample script uses requests session in the Connect Proxy Server module to invoke the resolve request.

You can also look at other sample scripts that use urlib.request in the Connect Proxy Server module to invoke the call.

```
from connectproxyserver import ConnectProxyServer, ProxyProtocol
# Mapping between SampleApp API response fields to CounterACT properties
sampleapp_to_ct_props_map = {
        "state": "connect_sampleapp_state",
        "last_logged_in_user": "connect_sampleapp_last_logged_in_user",
        "mac_addresses": "connect_sampleapp_mac_addresses",
        "is_safe": "connect_sampleapp_is_safe",
        "id": "connect_sampleapp_id"
}

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
logging.debug("SampleApp resolve")
token_returned = params.get("connect_authorization_token")
# Check if we have valid auth token or not before processing.
if token_returned:
        # 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 = token_returned
        if "mac" in params:
                logging.debug(f'mac is: {params.get("mac")}')
                mac = '-'.join(params.get("mac")[i:i+2] for i in range(0,
12, 2))
                get_mac_url = url + "/devices/v2/macaddress/" + mac
```

```
                logging.debug(f"get mac url is: {get_mac_url}")
                device_headers = {"Content-Type": "application/json;
charset=utf-8", "Authorization": "Bearer " + str(jwt_token)}

                try:
                        # Create proxy server
                        proxy_server = ConnectProxyServer(params)
                        # Pass to use what HTTPS or HTTP or both in the
protocol, pass down the ssl_verify
                        with
proxy_server.get_requests_session(ProxyProtocol.all,
headers=device_headers, verify=ssl_verify) as session:
                                # Example to show app writer can see what
is set in the session to debug
                                logging.debug(f"Headers is:
{session.headers}")
                                logging.debug(f"Session proxies is:
{session.proxies}")
                                logging.debug(f"Proxies:
{proxy_server.proxies}")
                                logging.debug(f"Verify is:
{session.verify}")
                                resolve_response =
session.get(get_mac_url, proxies=proxy_server.proxies)
                                logging.debug(f"Resolve response code:
{resolve_response.status_code}")
                                if 200 == resolve_response.status_code:
                                        logging.debug(f"Resolve response
text: {resolve_response.text}")
                                        request_response =
json.loads(resolve_response.text)
                                        """
                                        # 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 property
                                        """
                                        properties = {}
                                        if request_response:
                                                return_values =
request_response[0]
```

```
                                                    logging.debug(f"Resolve
response text on 0 element: {request_response[0]}")
                                                    for key, value in
return_values.items():
                                                            if key in
sampleapp_to_ct_props_map:

properties[sampleapp_to_ct_props_map[key]] = value

                                            response["properties"] = properties
                                    else:
                                            response["error"] =
resolve_response.reason
                        except Exception as e:
                                response["error"] = f"Could not resolve
properties: {e}."
                else:
                        response["error"] = "No mac address to query the endpoint
for."
else:
        response["error"] = "Unauthorized"
```

Sample app instance cache script for Connect

This topic provides a sample app instance cache script.

The following sample script uses urllib.request in the Connect Proxy Server module to invoke the call.

You can also look at other sample scripts that use requests session in the Connect Proxy Server module to invoke the call.

```
import urllib.request
from connectproxyserver import ConnectProxyServer, ProxyProtocol
from urllib.request import HTTPError, URLError

# CONFIGURATION
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
logging.debug("SampleApp instance cache for user")
# Check if we have valid auth token or not before processing.
if params.get("connect_authorization_token"):
    # ***** PART 2 - QUERY FOR USERS  ***** #
    jwt_token = params.get("connect_authorization_token")
```

```
    try:
        get_users_url = url + "/users/v2/"
        device_headers = {"Content-Type": "application/json;
charset=utf-8",
                          "Authorization": "Bearer " + str(jwt_token)}

        # Create proxy server
        proxy_server = ConnectProxyServer(params)
        https_handler = urllib.request.HTTPSHandler(context=ssl_context)
        # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_context
        opener = proxy_server.get_urllib_request_opener(ProxyProtocol.all,
https_handler)
        # Get users to save as app instance cache
        get_user_request = urllib.request.Request(get_users_url,
headers=device_headers)
        # Use urlopen
        get_user_response = urllib.request.urlopen(get_user_request)
        request_response = json.loads(get_user_response.read())
        logging.debug("Response json:
{}".format(json.dumps(request_response)))
        response_obj = {}

        for user_data in request_response.get("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"] = f"Could not connect to SampleApp. Response
code: {e.code}."
    except URLError as e:
        response["error"] = f"Could not connect to SampleApp. {e.reason}."
    except Exception as e:
        response["error"] = f"Could not connect to SampleApp. {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 handler token found."
```

Sample add a user action script for Connect

This topic provides a sample add a user action script.

The following sample script uses requests session in the Connect Proxy Server module to invoke the request.

You can also look at other sample scripts that use urlib.request in the Connect Proxy Server module to invoke the call.

```
import time
from connectproxyserver import ConnectProxyServer, ProxyProtocol

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
properties = {}
action_status = {}
# Example of check what proxy is set for the session
logging.debug("SampleApp add user")
# Check if we have valid auth token or not before processing.
if params.get("connect_authorization_token"):
    # ***** Execute add user action  ***** #
    jwt_token = params.get("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.get("sampleapp_email")
    if user_list 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_sampleapp_add_user_action"] = action_status
        response["properties"] = properties
    else:
        add_user_url = url + "/users/v2/"
        device_headers = {"Content-Type": "application/json;
charset=utf-8",
                          "Authorization": "Bearer " + str(jwt_token)}
        body = dict()
        logging.debug(f"Add user url: {add_user_url}")

        # 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.
        zones = {
            "id": "0927bf62-83f4-4766-a825-0b5d2e9749d0",
            "role_type": "00000000-0000-0000-0000-000000000002",
            "role_name": "User"
        }
        zone_array = [zones]
        body = {
            "email": params.get("sampleapp_email"),
            "user_role": "00000000-0000-0000-0000-000000000001",
            "first_name": params.get("sampleapp_first_name"),
            "last_name": params.get("sampleapp_last_name"),
            "zones": zone_array
        }

        try:
            # Create proxy server
            proxy_server = ConnectProxyServer(params)
            # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_verify
            with proxy_server.get_requests_session(ProxyProtocol.all,
headers=device_headers, verify=ssl_verify) as session:
                # If body in json or dictionary, use "json". If encoding,
or not json format, use "data".
```

```
                add_response = session.post(add_user_url, json=body,
proxies=proxy_server.proxies, timeout=10)
                logging.debug("Add user response code: " +
str(add_response.status_code))
                # Created response code is 201
                if 201 == add_response.status_code:
                    # If response is in json, use response.json(),
otherwise, use response.text
                    logging.debug(f"Add user response text:
{add_response.json()}")
                    request_response = add_response.json()

                    # 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
                    id = request_response.get('id')
                    logging.debug(f"The cookie content is: {id}")
                    response["cookie"] = id
                    action_status["status"] = "Succeeded"
                    action_status["time"] = int(time.time())
                    properties["connect_sampleapp_add_user_action"] =
action_status
                    properties["connect_sampleapp_last_logged_in_user"] =
params.get("sampleapp_email")
                    response["properties"] = properties
                else:
                    response["succeeded"] = False
                    response["troubleshooting"] = f"Failed action.
Response code: {add_response.status_code}."
                    action_status["status"] = "Failed. API return code is
not successful."
                    action_status["time"] = int(time.time())
                    properties["connect_sampleapp_add_user_action"] =
action_status
                    response["properties"] = properties
        except Exception as e:
```

```
            response["troubleshooting"] = f"Failed action: {str(e)}"
            response["succeeded"] = False
            action_status["status"] = "Failed. Exception."
            action_status["time"] = int(time.time())
            properties["connect_sampleapp_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_sampleapp_add_user_action"] = action_status
    response["properties"] = properties
```

Sample delete a user action script for Connect

This topic provides a sample delete a user action script.

The following sample script uses requests session in the Connect Proxy Server module to invoke the request.

You can also look at other sample scripts that use urlib.request in the Connect Proxy Server module to invoke the call.

```
import time
from connectproxyserver import ConnectProxyServer, ProxyProtocol

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
properties = {}
action_status = {}
logging.debug("SampleApp delete user")
# Check if we have valid auth token or not before processing.
if params.get("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.get("connect_authorization_token")
        delete_user_url = url + "/users/v2/" + params.get("cookie")
        logging.debug(f"Delete user url: {delete_user_url}")
        device_headers = {"Authorization": "Bearer " + str(jwt_token)}
```

```
        try:
                # Create proxy server
                proxy_server = ConnectProxyServer(params)
                # Pass to use what HTTPS or HTTP or both in the protocol,
pass down the ssl_verify
                with proxy_server.get_requests_session(ProxyProtocol.all,
headers=device_headers, verify=ssl_verify) as session:
                        delete_response = session.delete(delete_user_url,
proxies=proxy_server.proxies)
                        logging.debug(f"Proxies: {proxy_server.proxies}")
                        logging.debug(f"Delete user response code:
{delete_response.status_code}")
                        if 200 == delete_response.status_code:
                                response["succeeded"] = True
                                action_status["status"] = "Succeeded"
                                action_status["time"] = int(time.time())

properties["connect_sampleapp_add_user_action"] = action_status
                                response["properties"] = properties
                        else:
                                response["troubleshooting"] = f"Failed
code: {delete_response.status_code}. Reason: {delete_response.reason}."
                                logging.debug(f'Delete failed reason:
{response["troubleshooting"]}')
                                response["succeeded"] = False
                                action_status["status"] = "Response is not
successful."
                                action_status["time"] = int(time.time())

properties["connect_sampleapp_add_user_action"] = action_status
                                response["properties"] = properties
        except Exception as e:
                response["troubleshooting"] = f"Failed delete action.
Exception: {str(e)}"
                logging.debug(f'Delete failed reason:
{response["troubleshooting"]}')
                response["succeeded"] = False
                action_status["status"] = "Failed. Exception."
                action_status["time"] = int(time.time())
                properties["connect_sampleapp_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_sampleapp_add_user_action"] = action_status
response["properties"] = properties
```

Sample authorization script for Connect

This topic provides a sample authorization script.

The following sample script uses requests session in the Connect Proxy Server module to invoke the request.

You can also look at other sample scripts that use urlib.request in the Connect Proxy Server module to invoke the call.

```
import jwt  # PyJWT version 1.6.1 as of the time of authoring
import uuid
from datetime import datetime, timedelta
from connectproxyserver import ConnectProxyServer, ProxyProtocol

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params.get("connect_sampleapp_url")  # Server URL
tenant = params.get("connect_sampleapp_tenant_id")  # Tenant ID
app = params.get("connect_sampleapp_application_id")  # Application ID
secret = params.get("connect_sampleapp_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"}
response = {"token": ""}
token_url = url + "/auth/v2/token"
logging.debug("SampleApp auth.")
try:
    proxy_server = ConnectProxyServer(params)
    logging.debug(f"Get token url is: {token_url}")
    # This will close the session when exiting the block.
    # If use session =, then the session can be reused from the connection
pool until close
    with proxy_server.get_requests_session(ProxyProtocol.all,
headers=headers, verify=ssl_verify) as session:
        # Example of check what proxy is set for the session
        logging.debug(f"Proxies: {proxy_server.proxies}")
        # Make post call to get token, set time out 10 seconds
        post_response = session.post(token_url, json=payload, timeout=10)

        # Can also write as the following as example
        # session = proxy_server.get_requests_session(ProxyProtocol.http)
        # session.headers = headers
        # post_response = session.post(token_url, json=payload,
verify=ssl_verify)

        # # Use urlopen example
        # proxy_server = ConnectProxyServer(params)
        # # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_context
        # opener =
proxy_server.get_urllib_request_opener(ProxyProtocol.https, ssl_context)
        # auth_request = urllib.request.Request(token_url, headers=headers,
        # data=bytes(json.dumps(payload), encoding="utf-8"))
        # # Use opener.open to get the request ( you can use urlopen as
well, look at resolve script
        # auth_response = urllib.request.urlopen(auth_request)
        # post_response = json.loads(auth_response.read().decode("utf-8"))
        # logging.debug("Response json:
{}".format(json.dumps(post_response)))
        # jwt_token = post_response.get('access_token')  # access_token to
be passed to GET request
        # response["token"] = jwt_token
```

```
        logging.debug(f"Authorize response code:
{post_response.status_code}")
        if 200 == post_response.status_code:
            logging.debug(f"Authorize response encoding:
{post_response.encoding}")
            # response.json() has the return info in json format. Can use
response.text for a string.
            response["token"] = post_response.json().get('access_token')
        else:
            response["error"] = post_response.reason
        #
        # one_proxy_server = ConnectProxyServer(params)
        # proxies = one_proxy_server.get_proxies(ProxyProtocol.all)
        # if one_proxy_server.is_enabled:
        #     res = requests.post(token_url, proxies=proxies,
headers=headers, json=payload, verify=ssl_verify)
        # else:
        #     res = requests.post(token_url, proxies=None,
headers=headers, json=payload, verify=ssl_verify)
        #
        # logging.debug(f"No lib call. Get response {res.text}")

except Exception as e:
    logging.debug(f"Authorize failed error: {str(e)}")
    response["error"] = f"Failed to get token: {str(e)}"
```

Sample IOC poll script for Connect

This topic provides a sample IOC poll script.

```
"""
IOC Polling script is used to update IOC data periodically from third-
party vendor through API calls.
Use your own scripts to construct API request and process the API response
to parse IOC information.
The  response dictionary must contain a dictionary or a list of dictionary
called "ioc" so that IOC data can be sent to IOC Scanner
The below polling script shows an example of how to poll IOC data from
Crowdstrike
through API GET request https://api.crowdstrike.com/iocs/combined/
indicator/v1
"""

from urllib import request
```

```
import logging
import json
from urllib.request import HTTPError, URLError
from connectproxyserver import ConnectProxyServer, ProxyProtocol

url        = params.get("connect_sampleapp_url")
client_id   = params.get("connect_sampleapp_application_id")
client_secret = params.get("connect_sampleapp_application_secret")
bearer_token = params.get("connect_authorization_token")

response = {}
ioc_infos = []
IOC_NAME = "name"
IOC_HASH = "hash"
IOC_HASH_TYPE = "hash_type"
IOC_FILE_NAME = "file_name"
IOC_PLATFORM = "platform"
IOC_SEVERITY = "severity"

ioc_property_mapping = {
    IOC_HASH : "value",
    IOC_HASH_TYPE : "type",
    IOC_SEVERITY : "severity",
    IOC_PLATFORM : "platforms"
}

#check if bearer token is available; if not, create API request to
retrieve it
if not bearer_token:
    auth_url = f"{url}/oauth2/token"
    logging.debug("Get authentication URL: {}".format(auth_url))
    auth_headers = { "Content-Type": "application/x-www-form-urlencoded;
charset=utf-8",
                     "accept": "application/json"}
    auth_post_data = f"client_id={client_id}&client_secret={client_secret}"
    try:
        # Create proxy server
        proxy_server = ConnectProxyServer(params)
        # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_context
        opener =
proxy_server.get_urllib_request_https_opener(ProxyProtocol.all,
ssl_context)
        auth_request = urllib.request.Request(auth_url, data
```

```
=auth_post_data.encode(), headers = auth_headers)
        auth_response = opener.open(auth_request)
        logging.debug("response code  : [%s]", auth_response.getcode())
        if auth_response.getcode() == 201:
            auth_response_dict = json.loads(auth_response.read())
            if auth_response_dict.get("access_token"):
                bearer_token = auth_response_dict["access_token"]
                logging.debug("Successfully retrieved Bearer token")
    except HTTPError as e:
        logging.debug(f"Failed to retrieve Bearer token. HTTP Error :
Could not connect to {auth_url}. Error code: {e.code}")
    except URLError as e:
        logging.debug(f"Failed to retrieve Bearer token. URL Error : Could
not connect to {auth_url}. {e.reason}")
    except Exception as e:
        logging.debug( f"Failed to retrieve Bearer token. Could not
connect to {auth_url}. {e}")


ioc_poll_base_url = f"{url}/iocs/combined/indicator/v1"
if bearer_token:
    try:
        pagination_completed = False
        ioc_poll_url = ioc_poll_base_url
        while not pagination_completed:
            logging.debug("IOC Polling URL: {}".format(ioc_poll_url))
            # Create proxy server
            proxy_server = ConnectProxyServer(params)
            # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_context
            opener =
proxy_server.get_urllib_request_https_opener(ProxyProtocol.all,
ssl_context)
            ioc_poll_headers = { "Content-Type": "application/json;
charset=utf-8", "Authorization": "Bearer " + str(bearer_token) }
            ioc_poll_request = request.Request(ioc_poll_url,
headers=ioc_poll_headers)
            ioc_poll_response = opener.open(ioc_poll_request)
            if ioc_poll_response.getcode() == 200:
                ioc_poll_response_dict =
json.loads(ioc_poll_response.read())
                if "resources" in ioc_poll_response_dict:
                    ioc_data_list = ioc_poll_response_dict["resources"]
                    for ioc_data in ioc_data_list:
```

```
                        if "metadata" in ioc_data and "original_filename"
in ioc_data["metadata"]:
                            ioc_info = {}
                            ioc_info[IOC_NAME] = ioc_data["metadata"]
["product_name"]
                            ioc_info[IOC_FILE_NAME] = ioc_data["metadata"]
["original_filename"]
                            for field in ioc_property_mapping:
                                if ioc_property_mapping[field] in ioc_data
and ioc_property_mapping[field]!="platforms":
                                    ioc_info[field] =
ioc_data[ioc_property_mapping[field]]
                                elif ioc_property_mapping[field] ==
"platforms":
                                    ioc_info[field] =
",".join(ioc_data[ioc_property_mapping[field]])
                            ioc_infos.append(ioc_info)
            total = int(ioc_poll_response_dict["meta"]["pagination"]
["total"])
            offset = int(ioc_poll_response_dict["meta"]["pagination"]
["offset"])
            if total <= offset:
                pagination_completed = True
            else:
                ioc_poll_url = f"{ioc_poll_base_url}?offset={offset}"
        """
         For IOC polling, the response dictionary must contain a
dictionary or a list of dictionary called "ioc",
         which contains IOC information (single IOC or multiple IOCs
respectively).
         Each IOC should be formatted as a dictionary with four mandatory
keys of "name", "file_name", "hash" and "severity".
         To get more details about "ioc" response format, please refer to
Connect help file.
         The full response object, for example would be:
         {"ioc": {
             "name": "IOC Test 1",
             "hash_type": "Sha256",
             "hash":
"4747d091f230bf409af6a42c32cca30d845541d2c3200bb275f0fa179bd5e999",
             "severity": "low",
             "file_name": "cscript.exe"
             }
         }
```

```
            or
            {"ioc":
                [
                    {
                    "name": "IOC Test 1",
                    "hash_type": "Sha256",
                    "hash":
"4747d091f230bf409af6a42c32cca30d845541d2c3200bb275f0fa179bd5e999",
                    "severity": "medium",
                    "file_name": "cscript.exe"
                    "file_size": 44
                    },
                    {
                    "name": "IOC Test 2",
                    "hash_type": "md5",
                    "hash": "2e8d4a6fedf90e265d2370a543b4fd2a",
                    "severity": "low",
                    "file_name": "cscript.exe",
                    "file_exists": {'file_name': 'cscript.exe', 'file_path': 'c:\
\windows\\system32\\'}
                    "cnc": ["8.8.8.8", "9.9.9.9"]
                    },
                ]
            }
            """
            response["ioc"] = ioc_infos
            logging.debug(f"response: {response}")
        except HTTPError as e:
            response["error"] = f"HTTP Error : Could not connect to
{ioc_poll_url}. Error code: {e.code}"
            logging.debug(f"HTTP Error : Could not connect to {ioc_poll_url}.
Error code: {e.code}")
        except URLError as e:
            response["error"] = f"URL Error : Could not connect to
{ioc_poll_url}. {e.reason}"
            logging.debug(f"URL Error : Could not connect to {ioc_poll_url}.
{e.reason}")
        except Exception as e:
            response["error"] = f"Could not connect to {ioc_poll_url}. {e}"
            logging.debug( f"Could not connect to {ioc_poll_url}. {e}")
    else:
```

```
    response["error"] = "Bearer token is not available or empty"
    logging.debug("Bearer token is not available or empty")
```

Sample IOC resolve script for Connect

This topic provides a sample IOC resolve script.

```
from connectproxyserver import ConnectProxyServer, ProxyProtocol

# Mapping between SampleApp API response fields to CounterACT properties
sampleapp_to_ct_props_map = {
    "host_name": "connect_sampleapp_host_name",
    "os_version": "connect_sampleapp_os_version"
}

# CONFIGURATION
# All server configuration fields will be available in the 'params'
dictionary.
url = params.get("connect_sampleapp_url")  # Server URL

response = {}
logging.debug("SampleApp IOC resolve")
token_returned = params.get("connect_authorization_token")
# Check if we have valid auth token or not before processing.
if token_returned:
    # 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 = token_returned
    if "mac" in params:
        logging.debug(f'mac is: {params.get("mac")}')
        mac = '-'.join(params.get("mac")[i:i+2] for i in range(0, 12, 2))
        get_mac_url = url + "/devices/v2/macaddress/" + mac
        logging.debug(f"get mac url is: {get_mac_url}")
        device_headers = {"Content-Type": "application/json;
charset=utf-8", "Authorization": "Bearer " + str(jwt_token)}

        try:
            # Create proxy server
            proxy_server = ConnectProxyServer(params)
            # Pass to use what HTTPS or HTTP or both in the protocol, pass
down the ssl_verify
```

```
            with proxy_server.get_requests_session(ProxyProtocol.all,
headers=device_headers, verify=ssl_verify) as session:
                # Example to show app writer can see what is set in the
session to debug
                logging.debug(f"Headers is: {session.headers}")
                logging.debug(f"Session proxies is: {session.proxies}")
                logging.debug(f"Proxies: {proxy_server.proxies}")
                logging.debug(f"Verify is: {session.verify}")
                resolve_response = session.get(get_mac_url,
proxies=proxy_server.proxies)
                logging.debug(f"Resolve response code:
{resolve_response.status_code}")
                if 200 == resolve_response.status_code:
                    logging.debug(f"Resolve response text:
{resolve_response.text}")
                    request_response = json.loads(resolve_response.text)
                    """
                    # 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 property
                    """
                    properties = {}
                    if request_response:
                        return_values = request_response[0]
                        logging.debug(f"Resolve response text on 0
element: {request_response[0]}")
                        for key, value in return_values.items():
                            if key in sampleapp_to_ct_props_map:
                                properties[sampleapp_to_ct_props_map[key]]
= value

                    response["properties"] = properties
                else:
                    response["error"] = resolve_response.reason
                """
                We also can poll IOC data from 3rd party and send it to
IOC scanner during resolve.
                Use your own codes to construct the requests to get IOC
data as well as
                process the request response for the final data to pass to
the response dictionary.
```

```
                To pass IOC data, the response dictionary must contain a
dictionary or a list of dictionary called "ioc",
                which contains IOC information (single IOC or multiple
IOCs).
                To get more details about "ioc" response format, please
refer to Connect help file.
                """
                # process data to construct a dictionary for IOC data or
                # a list of dictionaries for multiple IOCs and pass it to
"ioc" response
                # Below is an example of a single IOC data
                ioc_info = {
                    "date": 1621975392000,
                    "name": "IOC Test 1",
                    "hash": "852d67a27e454bd389fa7f02a8cbe23f",
                    "hash_type": "md5",
                    "platform": "none",
                    "file_name": "test_file_1.exe",
                    "file_size": 10,
                    "severity": "medium"
                }
                response["ioc"] = ioc_info
        except Exception as e:
            response["error"] = f"Could not resolve properties: {e}."
    else:
        response["error"] = "No mac address to query the endpoint for."
else:
    response["error"] = "Unauthorized"
```

Sample parse Syslog message script for Connect

This topic provides a sample parse syslog message script.

```
"""
This sample script shows how a Connect App can parse Syslog message into
correct IOC data format using regex
then send IOC data to IOC Scanner.
The Syslog message in this example replicates the Syslog message received
by Carbon Black.
Different Syslog source will send Syslog message in different format, the
app writer should
customize their codes to handle and process the syslog message
respectively.
To check and get any available syslog, use:
```

```
params.get("connect_syslog_message")
If available, params["connect_syslog_message"] will be in String format
Sample Syslog message used in this script (added line separators for easy
reading):
-------
Syslog message: <28>Dec 10 09:51:22 2020-12-10 09: 51:22 [2783] <warning>
reason=feed.storage.hit
type=event process_guid=00000002-0000-069c-01d6-cf1b37991278
segment_id=1607622682337 host='mliu-2k12'
comms_ip='10.100.6.111' interface_ip='10.100.6.111' sensor_id=2 feed_id=22
feed_name='attackframework'
report_id='565644' report_title='(Unknown)' ioc_type='query'
ioc_value='{"index_type":"events",
"search_query":"cb.urlver=1&q=%28%28cmdline%3A.ps%2A%20OR%20cmdline%3A.bat%
20OR%20cmdline%3A.py%20OR
%20cmdline%3A.cpl%20OR%20cmdline%3A.cmd%20OR%20cmdline%3A.com%20OR%20cmdlin
e%3A.lnk%20OR%20cmdline%
3A.reg%20OR%20cmdline%3Ascr%20OR%20cmdline%3A.vb%2A%20OR%20cmdline%3A.ws%2A
%20OR%20cmdline%3A.xsl%
29%20-process_name%3Acrashpad_handler%20OR%20-
process_name%3AChrome.exe%29"}' timestamp='1607622682.338'
start_time='2020-12-10T17:38:05.785Z' group='default group'
process_md5='2e8d4a6fedf90e265d2370a543b4fd2a'
process_sha256='4747d091f230bf409af6a42c32cca30d845541d2c3200bb275f0fa179bd
5e082' process_name='cscript.exe'
process_path='c:\windows\system32\cscript.exe'
last_update='2020-12-10T17:38:05.972Z' alliance_score_attackframework='1'
alliance_link_attackframework='https://attack.mitre.org/techniques/T1059/'
alliance_data_attackframework='565644'
alliance_updated_attackframework='2020-09-28T20:22:00.000Z'
-------
The response dictionary must contain a dictionary or a list of dictionary
called "ioc" so that IOC data can be sent to IOC Scanner
"""

import re
import hashlib
import random
from datetime import datetime

IOC_NAME = "name"
IOC_HASH = "hash"
IOC_HASH_TYPE = "hash_type"
IOC_FILE_NAME = "file_name"
```

```
IOC_FILE_SIZE = "file_size"
IOC_PLATFORM = "platform"
IOC_DATE = "date"
IOC_SEVERITY = "severity"
IOC_FILE_EXISTS = "file_exists"
IOC_FILE_PATH = "file_path"
IOC_CNC = "cnc"
IOC_DNS = "dns"
IOC_MUTEX = "mutex"
IOC_SERVICE = "service"
IOC_PROCESS = "process"
IOC_PROCESS_NAME = "process_name"
IOC_PROCESS_HASH = "process_hash"
IOC_PROCESS_HASH_TYPE = "process_hash_type"
IOC_REGISTRY = "registry"
IOC_REGISTRY_ELEMENT = "registry_element"
IOC_REGISTRY_DATA = "registry_data"

def remove_till_word(input, keyword):
    """
    search for a keyword from the text input and return the string from
the index after the keyword to the end
    Eg: remove_till_word("feed_name='attackframework' report_id='565644'",
"feed_name=")
    -> 'attackframework' report_id='565644'
    :param input: string, the text (source) to search from
    :param keyword: string, the search term/ keyword to look for
    :return: string, the string starting after the keyword to the end
    """
    result = ""
    index = input.find(keyword)
    if index<0:
        return result
    index+=len(keyword)
    result = input[index:]
    return result

def parse_syslog_message(ioc_raw):
    """
    At this stage, the input consists of key=value pairs, separated by
spaces.
    The value itself may include spaces (in which case it'll be surrounded
by single quotes (')),
    e.g.: file_version='6.1.7600.16471
```

```
(win7_gdr_oob_havtool(wmbla).090930-1630)'
    The value may not be surrounded by single quotes, in which case it
shouldn't contain spaces,
    e.g.: sensor_id=8
    It's also possible that the value is surrounded by squarely brackets
([,]),
    e.g.: group=['Default Group']
    the regex in re.compile is used to separate these values from the rest
of the input.
    :param ioc_raw: a string received from syslog describing the IOC
    :return: a dictionary mapping the fields listed in the ioc string to
their corresponding value
    """
    #keywords look for these keywords when parsing the ioc string
    keywords = ["reason", "type", "md5", "host", "sensor_id",
"watchlist_id", "watchlist_name", "timestamp",
                "first_seen", "group", "desc", "company_name",
"product_name", "product_version", "file_version", "signed",
                "ioc_type", "ioc_value", "process_path", "parent_path",
"target_path", "file_path", "comms_ip", "interface_ip",
                "local_ip", "feed_name", "process_md5", "remote_ip"]
    infection_dict = {}
    #get the value string for each keyword
    for keyword in keywords:
        #first, get the beginning of the value to end of the string
        rest = remove_till_word(ioc_raw, keyword + "=")
        if rest:
            #pattern of the value
            pattern= re.compile("([^\']\\S*|\'.+?\'|\\[.+?\\])\\s*")
            found = pattern.findall(rest)
            if found:
                #if the found value contains single quote, eg:
'some_value', remove the quote
                val = found[0].replace("'","")
                #add keyword and its value into infection_dict
                infection_dict[keyword] = val
    return infection_dict

def set_ioc_name(event, infection_props, ioc_dict):
    """
    get IOC name from the infection info dictionary parsing from syslog
message and set name for ioc
    :param event: "reason" key value of preliminary infection info
dictionary,
```

```
                    indicates what syslog message is about.
    :param infection_props: a dictionary of the preliminary infection info
parsing from syslog message
    :param ioc_dict:  final ioc data dictionary
    :return: no turn, set key-value of ioc name for ioc dictionary
    """
    ioc_name = ""
    if infection_props.get("feed_name"):
        ioc_name = infection_props.get("feed_name")
    elif event.startswith("watchlist.hit") and
infection_props.get("watchlist_name"):
        ioc_name = infection_props.get("watchlist_name")
    elif infection_props.get("type"):
        if infection_props.get("ioc_value") and not
infection_props.get("ioc_value").startswith("{"):
            ioc_name = infection_props.get("ioc_type") + " " +
infection_props.get("ioc_value")
        else:
            ioc_name = infection_props.get("ioc_type")
    if ioc_name:
        ioc_dict[IOC_NAME] = ioc_name

def set_file(infection_props, ioc_dict):
    """
    set file and file exists information for ioc. File information can be
stored in one of four keywords
    including "process_path", "parent_path", "target_path" or "file_path"
in syslog message
    :param infection_props: a dictionary of the preliminary infection info
parsing from syslog message
    :param ioc_dict: final ioc data dictionary
    :return: no return. set key-value of file_name and file_exists for ioc
dictionary
    """
    file_name = ""
    file_path = ""
    for path_type in ["process_path", "parent_path", "target_path",
"file_path"]:
        if infection_props.get(path_type):
            file_info = get_file_info(infection_props[path_type])
            file_name = file_info[0]
            file_path = file_info[1]
            break
    if not file_name and infection_props.get("desc"):
```

```
            file_name = infection_props["desc"]
        if file_name and file_name not in ["//", ".//"]:
            ioc_dict[IOC_FILE_NAME] = file_name
            if file_path:
                ioc_file_exists = {
                    IOC_FILE_NAME : file_name,
                    IOC_FILE_PATH : file_path
                }
                ioc_dict[IOC_FILE_EXISTS] = ioc_file_exists

    def set_hash(infection_props, ioc_dict):
        """
        set hash information for ioc. For Carbon Black specifically, hash
    information can be stored in one of two keywords
        including "process_md5" or "md5" in syslog message
        :param infection_props: a dictionary of the preliminary infection info
    parsing from syslog message
        :param ioc_dict: final ioc data dictionary
        :return: no return. set key-value of hash and hash_type for ioc
    dictionary
        """
        #set hash type and hash value
        #Carbon black normally only sends md5 hash
        hash_value = ""
        hash_type = ""
        md5_keywords = ["process_md5", "md5"]
        for md5_keyword in md5_keywords:
            if infection_props.get(md5_keyword):
                hash_type = "md5"
                hash_value = infection_props[md5_keyword]
        if not hash_type:
            #generate random md5 hash and set type "none" if no hash available
            hash_type = "none"
            hash_value = generate_random_md5()
        ioc_dict[IOC_HASH_TYPE] = hash_type
        ioc_dict[IOC_HASH] = hash_value

    def set_severity(raw_ioc_data, ioc_dict):
        """
        set ioc severity. We use get_severity() to extract severity
    information from syslog message
        :param infection_props: a dictionary of the preliminary infection info
    parsing from syslog message
        :param ioc_dict: final ioc data dictionary
```

```
    :return: no return. set key-value of severity for ioc dictionary
    """
    ioc_severity = get_severity(raw_ioc_data)
    if ioc_severity:
        ioc_dict[IOC_SEVERITY] = ioc_severity

def set_cnc_dns(ioc_type, infection_props, ioc_dict):
    """
    set cnc or dns data for ioc
    :param ioc_type: type of ioc (cnc or dns)
    :param infection_props: a dictionary of the preliminary infection info
parsing from syslog message
    :param ioc_dict: final ioc data dictionary
    :return: no return. set key-value of cnc address or dns query for ioc
dictionary
    """
    if infection_props.get("ioc_type") and
infection_props.get("ioc_value") and infection_props["ioc_type"] ==
ioc_type:
        ioc_type_value = infection_props["ioc_value"]
        ioc_dict[ioc_type] = ioc_type_value

def set_service(infection_props, ioc_dict):
    """
    set service information for ioc
    :param infection_props: a dictionary of the preliminary infection info
parsing from syslog message
    :param ioc_dict: final ioc data dictionary
    :return:  no return. set key-value of service for ioc dictionary
    """
    ioc_service = ""
    if infection_props.get("product_name"):
        ioc_service = infection_props["product_name"]
    if infection_props.get("product_version"):
        ioc_service += infection_props["product_version"]
    if ioc_service:
        ioc_dict[IOC_SERVICE] = ioc_service

def set_platform(ioc_dict):
    #set platform for ioc
    #Some 3rd parties don't normally report the OS (eg: Carbon Black)
    ioc_dict[IOC_PLATFORM] = "none"

def get_file_info(file_path):
```

```
    """
    get the separate file name and file path information by splitting the
whole path using the last "\" character
    :param file_path: complete file path from which we extract the last
part, i.e. the file name
    :return: a list of file name and file path
    """
    pos = file_path.rfind("\\")
    if pos>0:
        file_name = file_path[pos+1:]
        path_name = file_path[:pos+1]
        return [file_name, path_name]
    return [file_path, file_path]

def get_severity(infection):
    """
    extract severity score from the raw data then map its value to
severity category
    :param infection: the raw infection data or syslog message
    :return: string, severity category of the infection
    """
    LOW_SEVERITY_SCORE = 25
    MEDIUM_SEVERITY_SCORE = 50
    HIGH_SEVERITY_SCORE = 75
    LOW_SEVERITY = "low"
    MEDIUM_SEVERITY = "medium"
    HIGH_SEVERITY = "high"
    CRITICAL_SEVERITY = "critical"
    feed_score_name = "alliance_score_"
    severity = ""
    score_index = infection.find(feed_score_name)
    if score_index > 0:
        score_value_index = infection.find("=", score_index)
        rest = infection[score_value_index + 1 :]
        pattern = re.compile("([^\']\\S*|\'.+?\'|\\[.+?\\])\\s*")
        found = pattern.findall(rest)
        if found:
            val = found[0].replace("\\","").replace("'","")
            score = int(val)
            if score < 0:
                return severity
            elif score <= LOW_SEVERITY_SCORE:
                severity = LOW_SEVERITY
            elif score <= MEDIUM_SEVERITY_SCORE:
```

```
                severity = MEDIUM_SEVERITY
            elif score <= HIGH_SEVERITY_SCORE:
                severity = HIGH_SEVERITY
            else:
                severity = CRITICAL_SEVERITY
    return severity

def get_ip_address(infection_props):
    """
    extract ipv4 address value from infection_props dictionary
    :param infection_props: a dictionary of the preliminary infection info
parsing from syslog message
    :return: ipv4 address in string format, empty string if no ip available
    """
    ipv4_address = ""
    if infection_props.get("comms_ip"):
        ipv4_address = infection_props["comms_ip"]
    elif infection_props.get("interface_ip"):
        ipv4_address = infection_props["interface_ip"]
    elif infection_props.get("local_ip"):
        ipv4_address = infection_props["local_ip"]
    return ipv4_address

def generate_random_md5():
    #generate a random md5 hash
    m = hashlib.md5(get_salt_string().encode())
    return m.hexdigest()

def get_salt_string():
    salt_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
    salt_list = []
    #create a random md5 hash containing 18 alphanumeric characters
    while len(salt_list)<18:
        random_index = random.randint(len(salt_chars))
        salt_list.append(salt_chars[random_index])
    return "".join(salt_list)

def get_ioc_infos(ioc_raw, infection_props):
    #get preliminary infection info as a dictionary from raw syslog message
    ioc_infos = {}
    #"reason" key value indicates what syslog message is about.
    # If the "reason" value starts with "feed.", "watchlist.hit" or
"binaryinfo", we know that syslog message contains IOC information
    event = infection_props.get("reason")
```

```
    if event and (event.startswith("feed.") or
event.startswith("watchlist.hit") or event.startswith("binaryinfo")):
        #set ioc name
        set_ioc_name(event, infection_props, ioc_infos)
        #set platform
        set_platform(ioc_infos)
        #set file and file exists
        set_file(infection_props, ioc_infos)
        #set ioc severity
        set_severity(ioc_raw, ioc_infos)
        #set hash
        set_hash(infection_props, ioc_infos)
        #set ioc type - cnc
        set_cnc_dns(IOC_CNC, infection_props, ioc_infos)
        #set ioc type - dns
        set_cnc_dns(IOC_DNS, infection_props, ioc_infos)
        #set ioc type - service
        set_service(infection_props, ioc_infos)
    return ioc_infos

def convert_time_str_to_epoch_num(time_str):
    """
    Convert time from string type to epoch number
    Args: str
        time_str: time in string type
    Returns:
        epoch number in int type if the time string is in correct format
(mm/dd/yyyy hh:mm:ss) else None
    """
    try:
        epoch = int(datetime.strptime(time_str, "yyyy-MM-
dd'T'HH:mm:ss.SSS'Z'").timestamp())
        logging.debug('Time string "{}" converted to epoch
successfully'.format(time_str))
        return epoch
    except ValueError as e:
        logging.info('Time string is: {}. {}'.format(time_str, e))
        return None

response = {}
properties = {}
logging.debug("SampleApp parse syslog message.")
ioc_property_mapping = {
    IOC_NAME : "name",
```

```
    IOC_DATE : "date",
    IOC_FILE_NAME : "file_name",
    IOC_HASH : "hash",
    IOC_HASH_TYPE : "hash_type",
    IOC_SEVERITY : "severity"
}
# Check if we have syslog message or not before processing.
if params.get("connect_syslog_message"):
    syslog_message = params.get("connect_syslog_message")
    logging.debug(f"Syslog message is: {syslog_message}")
    try:
        # Do parsing
        infection_props = parse_syslog_message(syslog_message)
        ioc_info_dict = get_ioc_infos(syslog_message, infection_props)
        ip_address = get_ip_address(infection_props)
        """
        To resolve properties, the response dictionary must contain a
dictionary called "properties"
        and a string of "mac" or "ip" or both (in case the app writer
wants to resolve with mac as a key and ip as a property)
        """
        threat_info = {}
        for key in ioc_property_mapping:
            if key == IOC_DATE and key in ioc_info_dict:
                threat_info[ioc_property_mapping[key]] =
convert_time_str_to_epoch_num(ioc_info_dict[key])
            elif key in ioc_info_dict:
                threat_info[ioc_property_mapping[key]] = ioc_info_dict[key]
        # if no date available, set date as current date
        if ioc_property_mapping[IOC_DATE] not in threat_info:
            threat_info[ioc_property_mapping[IOC_DATE]] =
int(datetime.now().timestamp())
        properties["connect_sampleapp_threat_info"] = [threat_info]
        response["properties"] = properties
        response["ip"] = ip_address
        """
        To send IOC data to IOC Scanner, the response dictionary must
contain
        a dictionary or a list of dictionary called "ioc",
        which contains IOC information (single IOC or multiple IOCs
respectively).
        Each IOC should be formatted as a dictionary with four mandatory
keys of "name", "file_name", "hash" and "severity".
        To get more details about "ioc" response format, please refer to
```

```
Connect help file.
        """
        response["ioc"] = ioc_info_dict
        logging.debug("response: {}".format(response))
    except Exception as e:
        response["error"] = f"Could not connect to SampleApp. {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["error"] = "No syslog message is passed."
```

Appendix B: Connect Proxy Server module

This topic describes the Connect Proxy Server module.

The Connect Proxy Server module is a Python module for handling proxy server calls to support the Proxy Server panel. Import connectproxyserver module in Python scripts and use corresponding connectproxyserver methods to call requests session or urllib.request.

To leverage the connectproxyserver module and predefined Proxy Server panel:

1. Specify "`proxy server": true` in the system.conf file. This leverages the predefined Proxy Server panel. You can go to the Connect user interface to enable the proxy server by selecting the Use Proxy Server checkbox and then entering the other fields in the Proxy Server panel. For Proxy Server details, see Proxy Server Panel Details.
2. In Python scripts:
   a. Import the connectproxyserver module by specifying:

      ```
      from connectproxyserver import ConnectProxyServer, ProxyProtocol
      ```

   b. Create connectproxyserver instance from class call:

      ```
      proxyserver = ConnectProxyServer(params)
      ```

   c. Use methods in connnectproxyserver to either make urlib.request call or requests session call.

For an example using urllib.request, see the following in Sample Polling Script for Connect:

```
opener = proxy_server.get_urllib_request_https_opener(ProxyProtocol.all,
ssl_context)

poll_request = urllib.request.Request(get_mac_url, headers=device_headers)
```

For an example using requests session, see the following in Sample Resolve Script for Connect:

```
with proxy_server.get_requests_session(ProxyProtocol.all,
headers=device_headers, verify=ssl_verify) as session:
```

For all script examples, see the Sample Connect Script Files.

More information is available than in this appendix, such as source code and usage examples. Refer to the Connect-Library folder on GitHub for details:

https://github.com/Forescout/eyeExtend-Connect/tree/master/Connect-Library

Connect Proxy Server classes

This topic describes Proxy Server classes.

There are two Proxy Server classes available: ConnectProxyServer and ProxyProtocol.

ConnectProxyServer

```
class ConnectProxyServer(params)
```

This class takes parameters as input and setup proxy server info if there is any in the constructor. You can use class methods that 1) use requests or 2) use urllib.request to invoke http requests. If proxy server is enabled, both the proxy IP and proxy port are required, in this case, if not specified, ValueError is raised. Username and password for the proxy server are optional. Only when the proxy server is enabled, which is specified in params.get("connect_proxy_enable"), proxy server info is used in the method calls. When constructed, the self.proxies is None.

:keyword params, variables passed by the Connect script:

- connect_proxy_enable, if not specified or false, indicates no proxy server.
- connect_proxy_ip, required when connect_proxy_enable is true
- connect_proxy_port, required when connect_proxy_enabled is true
- connect_proxy_username, optional
- connect_proxy_password, optional

:raise ValueError if proxy server is enabled and proxy IP or proxy port is not specified.

Methods

This topic describes the ConnectProxyServer methods.

get_requests_session

```
def get_requests_session(self, protocol=ProxyProtocol.https, **kwargs)
```

Get a requests session with proxy server information set.

Method will check if the proxy is enabled or not. And after getting the requests session, you can use post, delete, put (whatever the requests session supports) to invoke a http request.

By using this method, user proxy server is transparent to the user. The proxy server configuration is set. Refer to sessions to see the variables that can be passed:

https://requests.readthedocs.io/en/master/api/#request-sessions

:param self:

:param protocol: Proxies to use in the session for the proxy server, if proxy server is enabled, can be ProxyProtocol.https, ProxyProtocol.http, or ProxyProtocol.none. The default is ProxyProtocol.https. This sets the session proxies, auth in HTTPProxyAuth and trust_env.

:param kwargs: (optional) Any arguments that requests.session supports, except proxies, auth, and trust_env, passed as "headers=headerValue", for example. You can also pass the value via session methods such as post, get, delete etc. You can also pass use_cache=true which will persist in the connection pool. By default, we do not cache the connection in the connection pool.

:return: A requests.session that has proxy server configured.

get_urllib_request_https_opener

```
def get_urllib_request_https_opener(self, protocol=ProxyProtocol.https,
ssl_context=None, basic_auth=None)
```

Get a urllib request opener with proxy server configured with ssl_context and HTTP basic auth handler set.

Similar to method get_urllib_request_opener, if you want methods to set ssl_context and basic auth. The method has HTTPSHandler with SSL context and HTTPBasicAuthHandler set for the opener. You can use opener.open or urllib.request.urlopen after this method to connect to invoke a http request. Refer to HTTPBasicAuthHandler:

https://docs.python.org/3/library/urllib.request.html#urllib.request.HTTPPasswordMgr

:param protocol: Proxies to use in the session for the proxy server, if proxy server is enabled, can be ProxyProtocol.https, ProxyProtocol.http, or ProxyProtocol.none. The default is ProxyProtocol.https.

:param ssl_context: ssl_context passed to the context in the HTTPSHandler. The default is None.

:param basic_auth: HTTPPasswordMgr or similar passed to create HTTPBasicAuthHandler. The default is None.

:return: opener: OpenerDirector from urllib.request.build_opener that has ProxyHandler, HTTPSHandler with SSL context, and HTTPBasicAuthHandler set. The subsequent request call can use opener.open or urllib.request.urlopen.

get_urllib_request_opener

```
def get_urllib_request_opener(self, protocol=ProxyProtocol.https, *handlers)
```

Get a urllib request opener with proxy server configured.

Get a request opener that can has proxy server set. You can use opener.open or urllib.request.urlopen after this method to connect to invoke a http request. Refer to:

https://docs.python.org/3/library/urllib.request.html#urllib.request.build_opener

:param protocol: Proxies to use in the request for the proxy server, if proxy server is enabled, can be ProxyProtocol.https, ProxyProtocol.http, or ProxyProtocol.none. The default is ProxyProtocol.https.

:param handlers: (optional) Handlers that urllib.request.build_opener accepts, such as HTTPSHandler.

:return: opener: OpenerDirector from urllib.request.build_opener that has ProxyHandler set. The subsequent request call can use opener.open or urllib.request.urlopen.

ProxyProtocol

```
class ProxyProtocol(value, names=None, *, module=None, qualname=None,
type=None, start=1)
```

Choices for proxy server proxies.

Indicate the proxy server proxies (string) that are used in the request. Can be all, http, https, or none. If proxy server supports both protocols, can use ProxyProtocol.all. It is a good practice to use ProxyProtocol.all if supporting both protocols. If only supporting http, use ProxyProtocol.http. If supporting https, then use ProxyProtocol.https.

Ancestors:

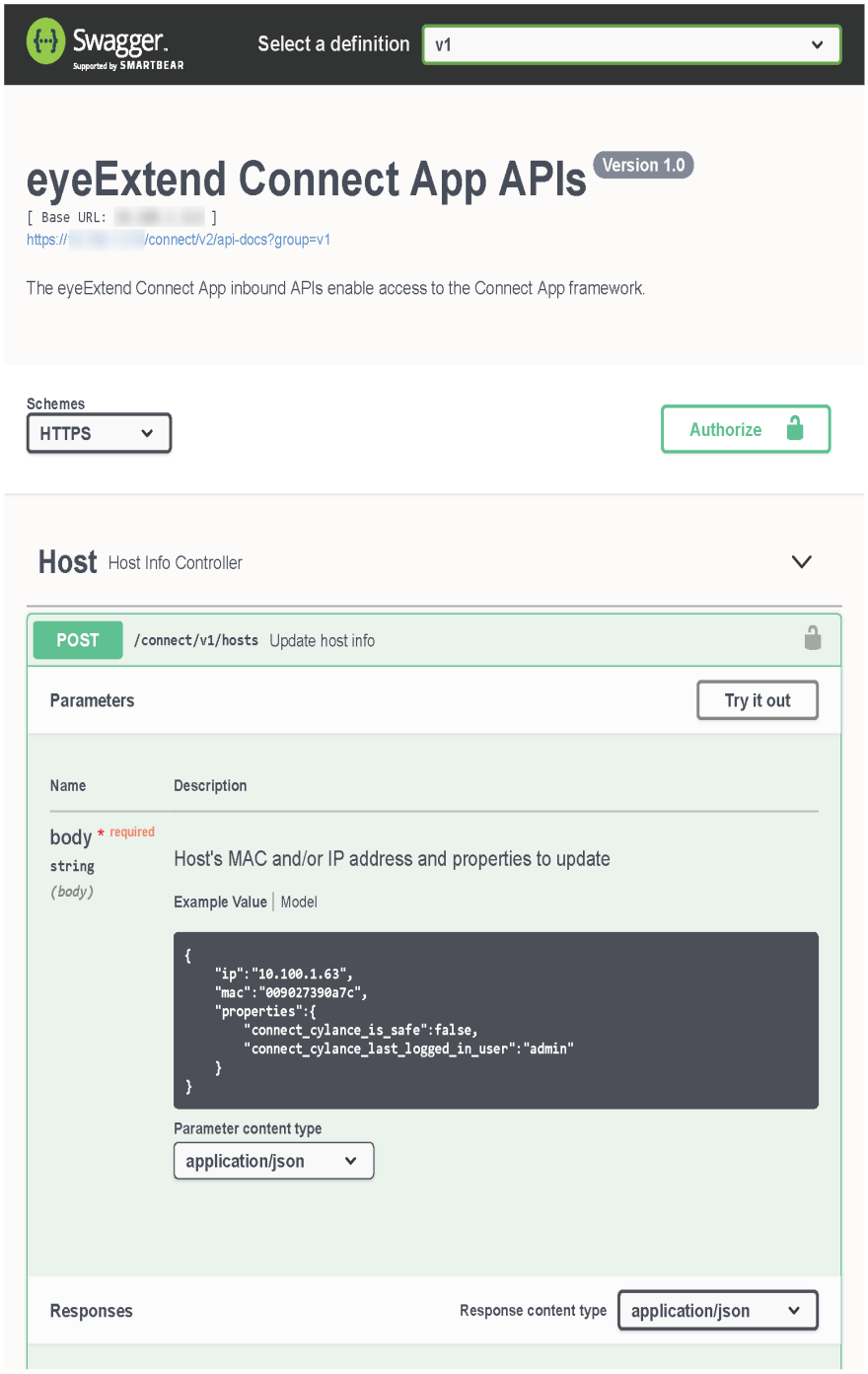
- enum.Enum

Class variables:

- var all
- var http
- var https
- var none

Appendix C: Swagger user interface

This topic provides a static version of the Swagger UI.


Swagger
Supported by SMARTBEAR
Select a definition
v1
eyeExtend Connect App APIs
Version 1.0
[Base URL:]
https://[redacted]/connect/v2/api-docs?group=v1
The eyeExtend Connect App inbound APIs enable access to the Connect App framework.
Schemes
HTTPS
Authorize
Host
Host Info Controller
POST
/connect/v1/hosts
Update host info
Parameters
Try it out
Name
Description
body * required
string
(body)
Host's MAC and/or IP address and properties to update
Example Value | Model
{
"ip":"10.100.1.63",
"mac":"009027390a7c",
"properties":{
"connect_cylance_is_safe":false,
"connect_cylance_last_logged_in_user":"admin"
}
}
Parameter content type
application/json
Responses
Response content type
application/json

Code	Description
200	Request succeeded Example Value | Model

```
{
    "status":"OK",
    "code":200,
    "message":null,
    "data":{
        "hosts":[
            {
                "mac":"005056a83dfc",
                "ip":"1.1.1.1",
                "properties":{
                    "connect_cylance_is_safe":"true",
                    "connect_cylance_last_logged_in_user":"CA-S8-W7-1\\Administrator",
                }
            }
        ]
    }
}
```

400

Bad request. Request parameter or body may not in correct format. Check actual message.

Example Value |

```
{
    "status":"BAD_REQUEST",
    "code":400,
    "message":"Error message"
}
```

401

Unauthorized. JWT token provided is not authorized. Check actual message.

Example Value |

```
{
    "status":"UNAUTHORIZED",
    "code":401,
    "message":"Error message"
}
```

403

Forbidden. The app is not authorized in provided JWT token. Check actual message.

Example Value |

```
{
    "status":"FORBIDDEN",
    "code":403,
    "message":"Error message"
}
```


Code
Description
500
Internal server error. No expected response from Forescout. Check actual message.
Example Value
{
"status":"INTERNAL_SERVER_ERROR",
"code":500,
"message":"Error message"
}
GET
/connect/v1/hosts/{id} Get single host properties for a specified app
Parameters
Try it out
Name
Description
id * required
string
(path)
IP or MAC address
1.1.1.1 or 112233445566
properties
string
(query)
Names of properties to return
propertyName1,propertyName2
Responses
Response content type
application/json

Code	Description
200	Request succeeded Example Value | Model


```
{
    "status":"OK",
    "code":200,
    "message":null,
    "data":{
        "hosts":[
            {
                "mac":"005056a83dfc",
                "ip":"1.1.1.1",
                "properties":{
                    "connect_cylance_is_safe":"true",
                    "connect_cylance_last_logged_in_user":"CA-S8-W7-1\\Administrator",
                    "connect_cylance_id":"f5552f2f-d8d8-4e4b-b4a5-b95255253a23",
                    "connect_cylance_state":"Offline",
                    "connect_cylance_mac_addresses":["00-50-56-A8-3D-FC"]
                }
            }
        ]
    }
}
```


400	Bad request. Request parameter or body may not in correct format. Check actual message. Example Value |

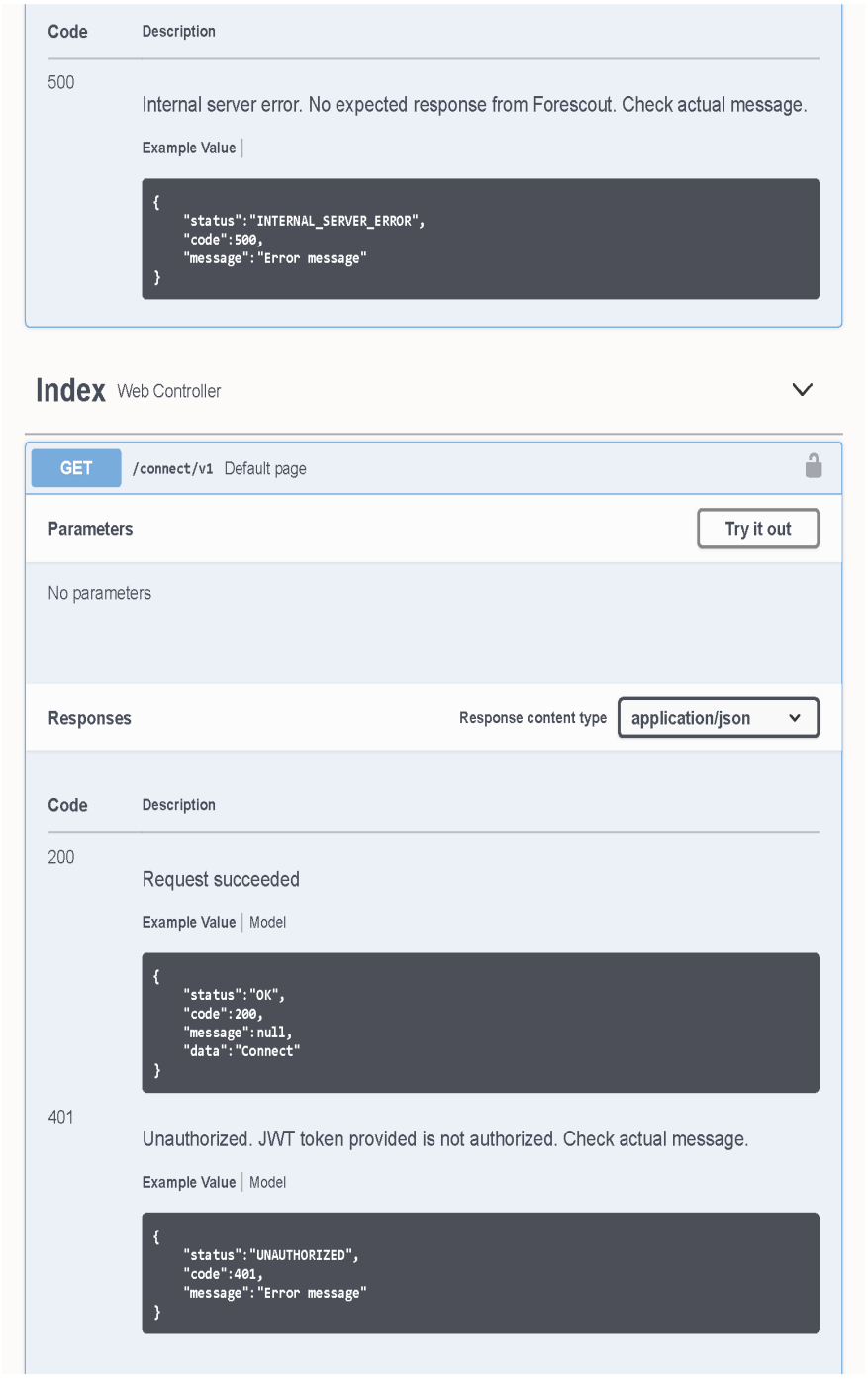

```
{
    "status":"BAD_REQUEST",
    "code":400,
    "message":"Error message"
}
```


401	Unauthorized. JWT token provided is not authorized. Check actual message. Example Value |


```
{
    "status":"UNAUTHORIZED",
    "code":401,
    "message":"Error message"
}
```


404	Not found. Host with this IP or MAC address is not found. Check actual message. Example Value |


```
{
    "status":"NOT_FOUND",
    "code":404,
    "message":"Error message"
}
```


Code
Description
500
Internal server error. No expected response from Forescout. Check actual message.
Example Value
{
"status":"INTERNAL_SERVER_ERROR",
"code":500,
"message":"Error message"
}
Index
Web Controller
GET
/connect/v1
Default page
Parameters
Try it out
No parameters
Responses
Response content type
application/json
Code
Description
200
Request succeeded
Example Value
Model
{
"status":"OK",
"code":200,
"message":null,
"data":"Connect"
}
401
Unauthorized. JWT token provided is not authorized. Check actual message.
Example Value
Model
{
"status":"UNAUTHORIZED",
"code":401,
"message":"Error message"
}


Code
Description
500
Internal server error. No expected response from Forescout. Check actual message.
Example Value | Model
{
"status":"INTERNAL_SERVER_ERROR",
"code":500,
"message":"Error message"
}
JWT Token
POST
/connect/v1/authentication/token
Get a JWT token
Method to get JWT token. After obtaining the token, copy it. Select the Authorize button and enter the token in format "Bearer ". The token is added to the header for all APIs.
Parameters
Try it out
Name
Description
body * required
string
(body)
Provide credentials and app name to obtain the JWT token. The "expiration" variable is optional and defaults to 15 minutes.
Example Value | Model
{
"username":"username",
"password":"password",
"app_name":"cylance",
"expiration":"15"
}
Parameter content type
application/json
Responses
Response content type
application/json

Code	Description

200	Successful response
	Example Value
	<pre>{
	"status":"200 OK",
	"code":"200",
	"message":"null",
	"data":"{
	"token":
	"eyJhbGciOiJIUzUxMiJ9.eyJhcHBfbmFtZSI6ImN5bGFuY2UiLCJzdWIiOiJhZG1pbiIsImlhdCI6MTYxMDM0NjgxMSwiZXhwIjoxNjEwMzQ3NzExLCJyb2wiOlsiUk9MRV9VU0VSIl19.iAxwATzEmCZavbLf1wNwTgvPwQ3nMix3dLwvlQawYBu0yppRmtr-dyA6LN7-s2nlZGDoFjjAiANcRnAIkPDRuA",
	"app_name": "cylance",
	"expire_time": 1610347711656
	}"
	}</pre>

400	Bad request
	Example Value
	<pre>{
	"status":"400 BAD_REQUEST",
	"code":"400",
	"message":"Error message"
	}</pre>

401	Failed to authorize
	Example Value
	<pre>{
	"status":"401 UNAUTHORIZED",
	"code":"401",
	"message":"Error message"
	}</pre>

500	Server failed to respond
	Example Value
	<pre>{
	"status":"500 INTERNAL_SERVER_ERROR",
	"code":"500",
	"message":"Error message"
	}</pre>

Models

ConnectApiResponse {

description: Response from a successful API call.

code integer($int32)

Status code of the API call. Use HttpStatus value.

data { description: Data returned in JSON format. Can be empty. }

message string

Message for the API call. Used commonly for error messages. Do not use this field to check if the API is successful.

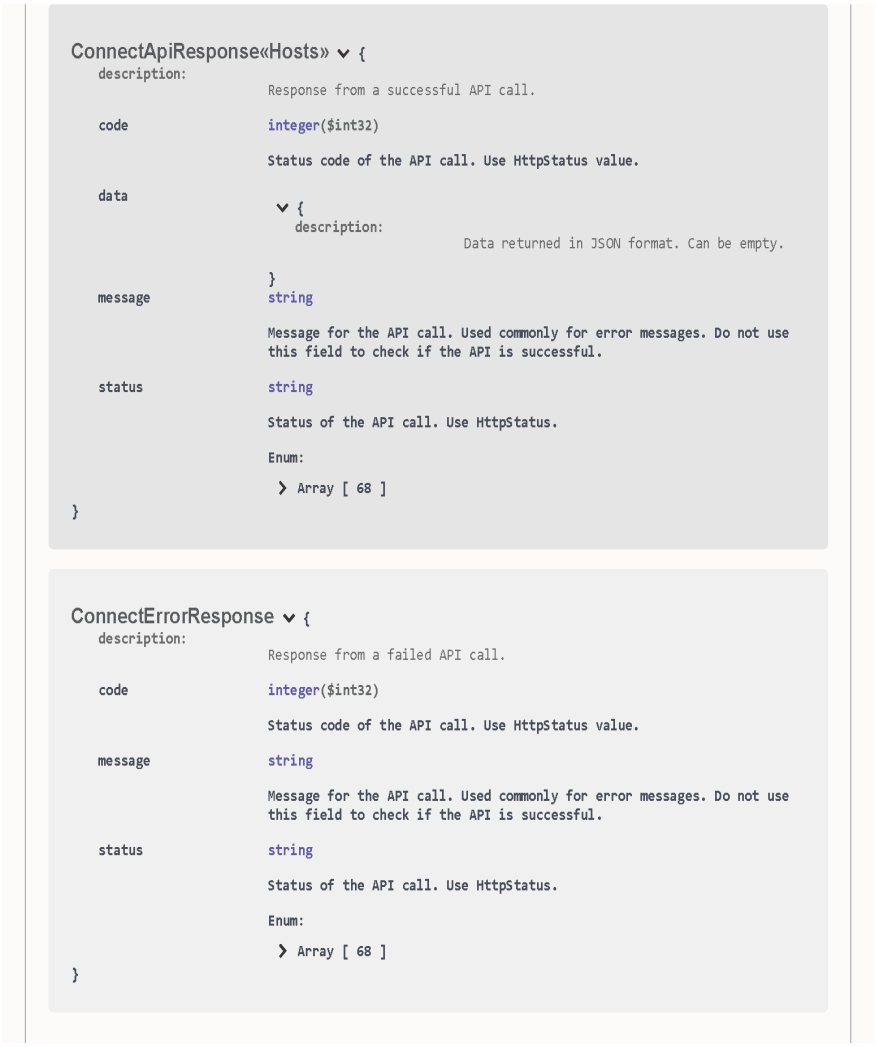
status string

Status of the API call. Use HttpStatus.

Enum:

[ACCEPTED, ALREADY_REPORTED, BAD_GATEWAY, BAD_REQUEST, BANDWIDTH_LIMIT_EXCEEDED, CHECKPOINT, CONFLICT, CONTINUE, CREATED, DESTINATION_LOCKED, EXPECTATION_FAILED, FAILED_DEPENDENCY, FORBIDDEN, FOUND, GATEWAY_TIMEOUT, GONE, HTTP_VERSION_NOT_SUPPORTED, IM_USED, INSUFFICIENT_SPACE_ON_RESOURCE, INSUFFICIENT_STORAGE, INTERNAL_SERVER_ERROR, I_AM_A_TEAPOT, LENGTH_REQUIRED, LOCKED, LOOP_DETECTED, METHOD_FAILURE, METHOD_NOT_ALLOWED, MOVED_PERMANENTLY, MOVED_TEMPORARILY, MULTIPLE_CHOICES, MULTI_STATUS, NETWORK_AUTHENTICATION_REQUIRED, NON_AUTHORITATIVE_INFORMATION, NOT_ACCEPTABLE, NOT_EXTENDED, NOT_FOUND, NOT_IMPLEMENTED, NOT_MODIFIED, NO_CONTENT, OK, PARTIAL_CONTENT, PAYLOAD_TOO_LARGE, PAYMENT_REQUIRED, PERMANENT_REDIRECT, PRECONDITION_FAILED, PRECONDITION_REQUIRED, PROCESSING, PROXY_AUTHENTICATION_REQUIRED, REQUESTED_RANGE_NOT_SATISFIABLE, REQUEST_ENTITY_TOO_LARGE, REQUEST_HEADER_FIELDS_TOO_LARGE, REQUEST_TIMEOUT, REQUEST_URI_TOO_LONG, RESET_CONTENT, SEE_OTHER, SERVICE_UNAVAILABLE, SWITCHING_PROTOCOLS, TEMPORARY_REDIRECT, TOO_EARLY, TOO_MANY_REQUESTS, UNAUTHORIZED, UNAVAILABLE_FOR_LEGAL_REASONS, UNPROCESSABLE_ENTITY, UNSUPPORTED_MEDIA_TYPE, UPGRADE_REQUIRED, URI_TOO_LONG, USE_PROXY, VARIANT_ALSO_NEGOTIATES]

}



ConnectApiResponse«Hosts» {

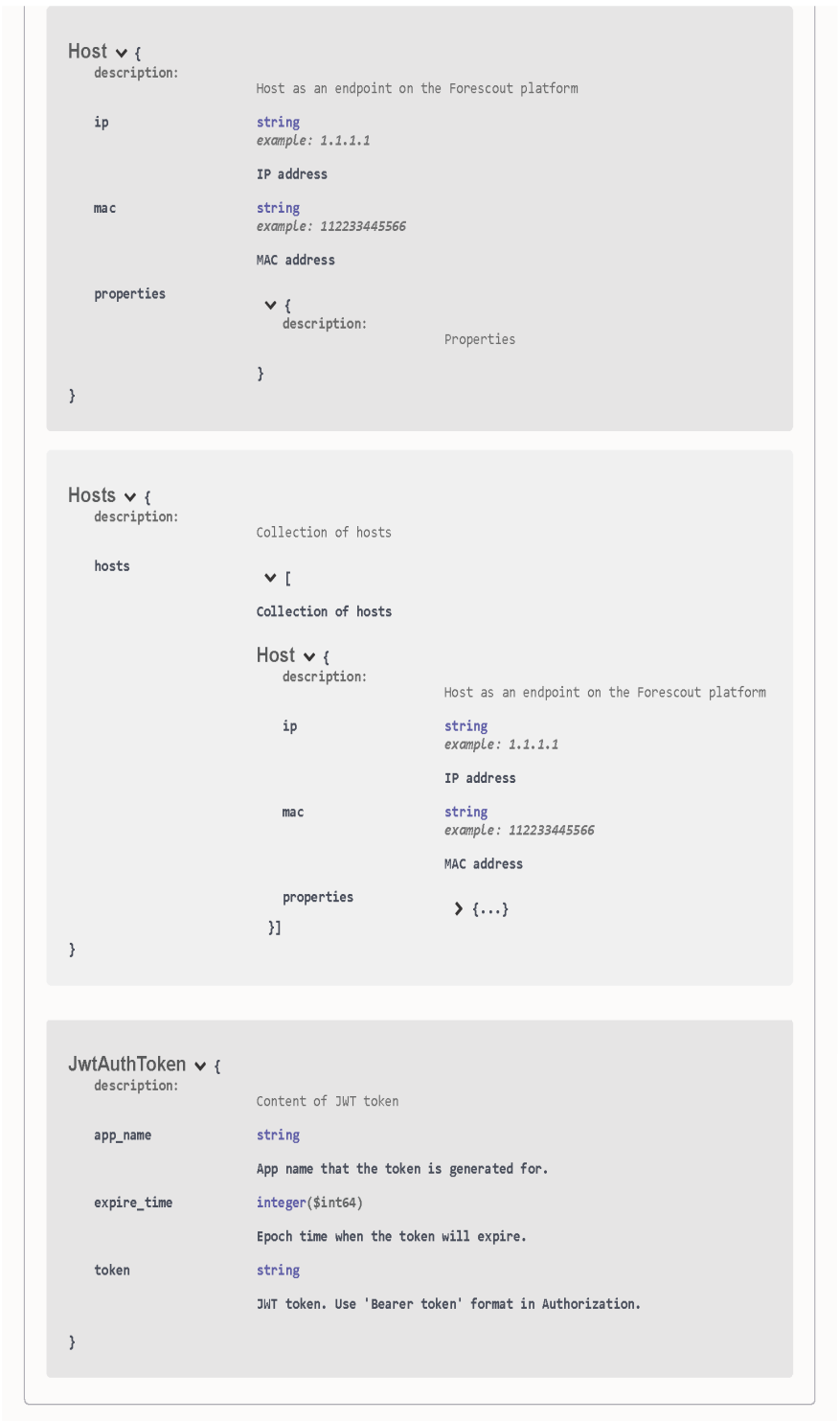
description:	Response from a successful API call.
code	integer($int32) Status code of the API call. Use HttpStatus value.
data	{ description: Data returned in JSON format. Can be empty. }
message	string Message for the API call. Used commonly for error messages. Do not use this field to check if the API is successful.
status	string Status of the API call. Use HttpStatus. Enum: Array [68]

}

ConnectErrorResponse {

description:	Response from a failed API call.
code	integer($int32) Status code of the API call. Use HttpStatus value.
message	string Message for the API call. Used commonly for error messages. Do not use this field to check if the API is successful.
status	string Status of the API call. Use HttpStatus. Enum: Array [68]

}



Host {
description: Host as an endpoint on the Forescout platform
ip string example: 1.1.1.1 IP address
mac string example: 112233445566 MAC address
properties { description: Properties }
}

Hosts {
description: Collection of hosts
hosts [Collection of hosts
Host {
description: Host as an endpoint on the Forescout platform
ip string example: 1.1.1.1 IP address
mac string example: 112233445566 MAC address
properties {...} }]
}

JwtAuthToken {
description: Content of JWT token
app_name string App name that the token is generated for.
expire_time integer($int64) Epoch time when the token will expire.
token string JWT token. Use 'Bearer token' format in Authorization.
}