

## Studio guide

---

ForgeRock® Identity Platform serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see <https://www.forgerock.com> .

This guide gives an overview of how to use IG Studio to design and develop routes to protect applications.

## Getting started with Studio

---

IG Studio is a user interface to help you build and deploy your IG configuration. There are two ways to create routes in Studio:

- With the *structured editor* to build simple routes by using predefined menus and templates. The structured editor presents valid options and default values as you add filters, decorators, and other objects to your configuration.
- With the *freeform designer* to design complex, multi-branched routes. Drag handlers and filters from a side bar onto the canvas to begin designing the route. The freeform designer helps you to visualize the chain, and track the path of requests, responses, and contexts.

After installation, IG is by default in production mode. The `/routes` endpoint is not exposed or accessible, and Studio is effectively disabled. To access Studio, switch to development mode as described in [Switching from production mode to development mode](#).

If you provide a custom `config.json`, include a main router named `_router`. If a custom `config.json` is not provided, IG includes this router by default.

When IG is installed and running in development mode, as described in [Getting started](#), access Studio on <http://ig.example.com:8080/openig/studio> . The **Routes** screen is displayed:

## Routes

Configure IG to protect your applications

+ Create a route

Import a route

Filter...

NAME	URL	STATUS
------	-----	--------

During IG upgrade, routes that were previously created in Studio are automatically transferred to the new version of IG. Where possible, IG replaces deprecated settings with the newer evolved setting. If IG needs additional information to upgrade the route, the route status becomes **Compatibility update required**. Select the route, and provide the requested information.

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

## Creating and editing routes in Structured Editor

The following sections describe basic tasks for creating and deploying routes in the structured editor of Studio:

### Creating simple routes

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **Structured** to use the structured editor.
2. Enter the URL of the application you want to protect, followed by a path condition to access the route. For example, enter `http://app.example.com:8081/my-basic-route`.

The route is created, and menus to add configuration objects to the route are displayed.

3. On the top-right of the screen, select **Display** to review the route.

A route similar to this is displayed, where the path condition is used for the route name:

```
{
  "name": "my-basic-route",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/my-basic-route')}",
  "handler": "ReverseProxyHandler"
}
```

## Changing the basic settings of a route

1. In Studio, select  **ROUTES**, and then select a route with the  icon.
2. On the top-right of the screen select  **Route settings**.
3. Using the on-screen hints for guidance, change the name, condition, or other features of the route, and save the changes.
4. On the top-right of the screen, select  and  **Display** to review the route.

## Adding configuration to a route

After creating a route in the structured editor, you can add filters, decorators, scripts, and other configuration to the route.

### *Add other configuration to a route*

1. In Studio, select  **ROUTES**, and then select a route with the  icon.
2. Select one of the configuration options, and follow the on-screen hints to select configuration settings.

For routes to test with the examples in the Gateway guide, see [Example routes created with Structured Editor](#).

### *Add other filters to a route*

Use this procedure to add any filter type to the configuration.

1. In Studio, select  **ROUTES**, and then select a route with the  icon.
2. Select  **Other filters** > **+ New filter** > **Other filter**.

3. In **Create filter**, select a filter type from the list, enter a name, and optionally enter a configuration for the filter.

**NOTE**

Studio checks that the JSON is valid, but doesn't check that the configuration of the filter is valid. If the filter configuration isn't valid, when you deploy the route it fails to load.

When you save, the filter is added to the list of other filters but is not added to the configuration.

4. Enable the filter to add it to the configuration.

If you disable the filter again, it is removed from the route's chain but the configuration is saved. Simply enable the filter again to add it back in the chain.

## Managing the route chain

The  **Chain** view lists the filters in the order that they appear in the configuration.

Some filters have a natural position in the chain. For example, so that an authenticated user is given the correct permissions, an authentication filter must come before an authorization filter. Similarly, so that an authorization token is transformed, an authorization filter always comes before a token transformation filter.

Other filters have a flexible position in the chain. For example, an AssignmentFilter can be used before a request is handled or after a response is handled.

When the position of a filter is fixed, it is automatically placed in the correct position in the chain; you cannot change the position. When the position of a filter is flexible, the  icon is displayed, and you can drag the filter into a different position in the chain.

Select  **Chain** to view and manage the filters in the chain as follows:

- When the  icon is displayed, drag a filter up or down the chain.
- select  to edit a filter.
- Select  **Realm Settings** to disable and remove a filter from the chain.

For information about chains, see [Chain](#).

## Deploying and undeploying routes

### *Deploy a Route*

1. In Studio, select  **ROUTES**, and then select a route created with the structured editor (with the  icon).

2. On the top-right of the screen, select **:** and **Display** to review the route.
3. If the route is okay, select **Deploy** to push the route to the IG configuration.

**IMPORTANT**

If the route configuration is not valid, or if a service that the route relies on, such as an AM service, is not available, the route fails to deploy.

If the route deploys successfully, **Deployed** is displayed, and the **Deploy** button is greyed out.

4. Check the `$HOME/.openig/config/routes` folder in your IG configuration to see that the route is there.

By default, routes are loaded automatically into the IG configuration. You don't need to stop and restart IG. For more information, see [Prevent the reload of routes](#).

5. Check the system log to confirm that the route was loaded successfully into the configuration. For information about logs, see [Managing logs](#).

### *Undeploy a Route*

1. In Studio, select **ROUTES** and then select a route with the status **Deployed**.
2. On the top-right of the screen, select **:** and **Undeploy**, and then confirm your request.

The route is removed from the IG configuration. On the Studio screen, the route status **Deployed** is no longer displayed, and the **Deploy** option is active again.

## Creating and editing routes in Freeform Designer

The following sections describe how to create a simple route in the freeform designer of Studio, and then add configuration to the route. For examples of routes created with the freeform designer that can be tested with the examples in the Gateway guide, see [Example routes created with Freeform Designer](#).

### Creating simple routes

1. In IG Studio, create a route:

- a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **Freeform** to use the freeform designer.
2. Select **Basic** to create a route from a blank template.
  3. Enter a URL for the application you want to protect, followed by a path condition to access the route. For example, enter `http://app.example.com:8081/my-basic-route`.

The route is displayed on the **Flow** tab of the canvas.

4. On the top-right of the screen, select **Display** to review the route.

```
{
  "name": "my-basic-route",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/my-basic-route')}",
  "handler": "ReverseProxyHandler",
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler"
    },
    {
      "type": "BaseUriDecorator",
      "name": "baseUri"
    },
    {
      "type": "TimerDecorator",
      "name": "timer",
      "config": {
        "timeUnit": "ms"
      }
    },
    {
      "type": "CaptureDecorator",
      "name": "capture",
      "config": {
        "captureEntity": false,
        "captureContext": false,
        "maxEntityLength": 524288
      }
    }
  ]
}
```

```
]
}
```

## Changing the basic settings of a route

1. Using the route created in [freeform.adoc#studio-create-route-ff](#), on the top-right of the screen select  **Route settings**.
2. Using the on-screen hints for guidance, change the name, condition, or other features of the route, and save the changes.
3. On the top-right of the screen, select  and  **Display** to review the route.

## Adding objects to a route heap

1. Using the route created in [freeform.adoc#studio-create-route-ff](#), select  **All Objects > Create Object**.
2. In **Node Type**, select an object type from the drop down list. For example, create an AmService object, using the following values:
  - **Name:** AmService-1
  - **URI:** `http://am.example.com:8088/openam/`
  - **Agent:**
    - **Agent:** ig-agent
    - **Password:** password

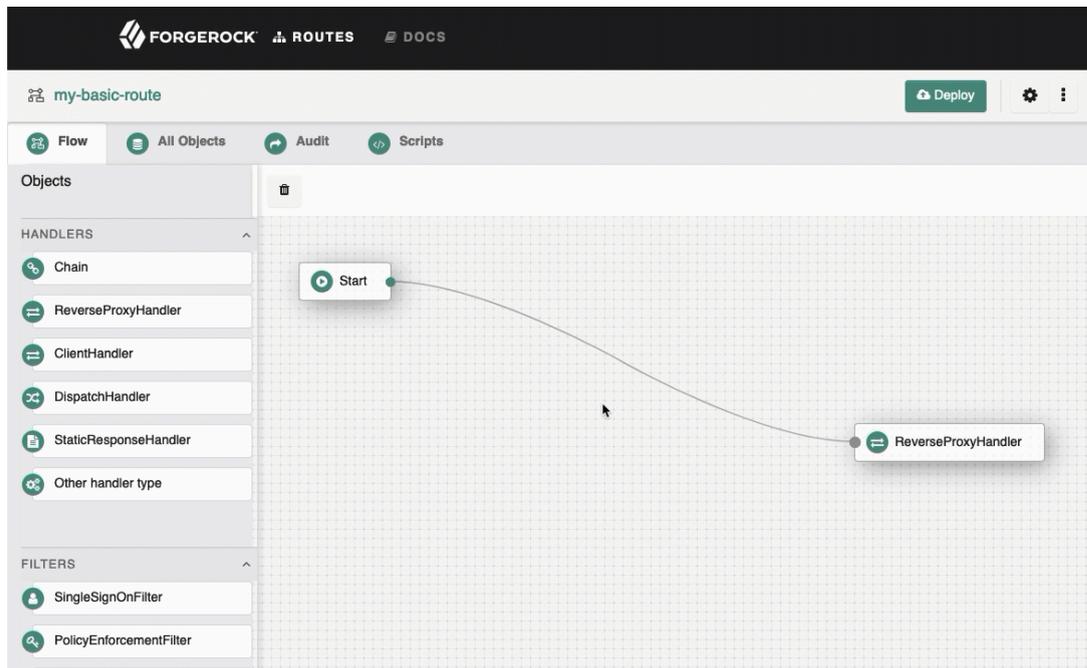
When you save, the object is added to route heap but is not used in the route.

3. On the top-right of the screen, select  and  **Display** to review the route.

## Adding configuration to a route

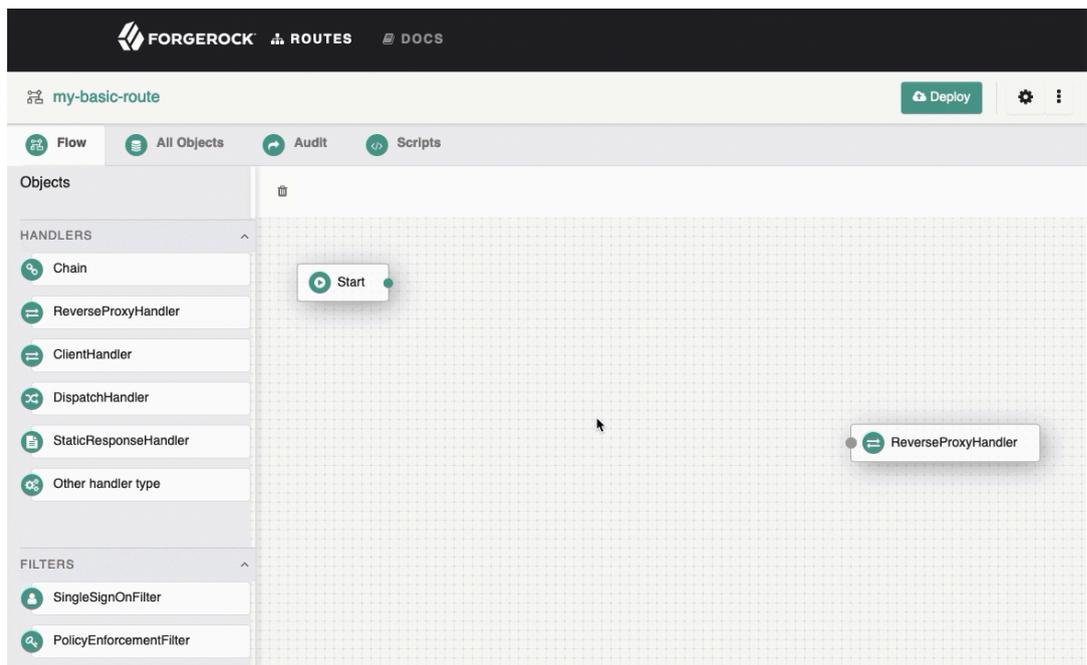
1. Using the route created in [freeform.adoc#studio-create-route-ff](#), select the  **Flow** tab, and delete the connector between **Start** and **ReverseProxyHandler**.

▼ [See how to delete a connector.](#)



2. Drag a **Chain** from the side bar onto the canvas, and then drag a **SingleSignOnFilter** into the chain.

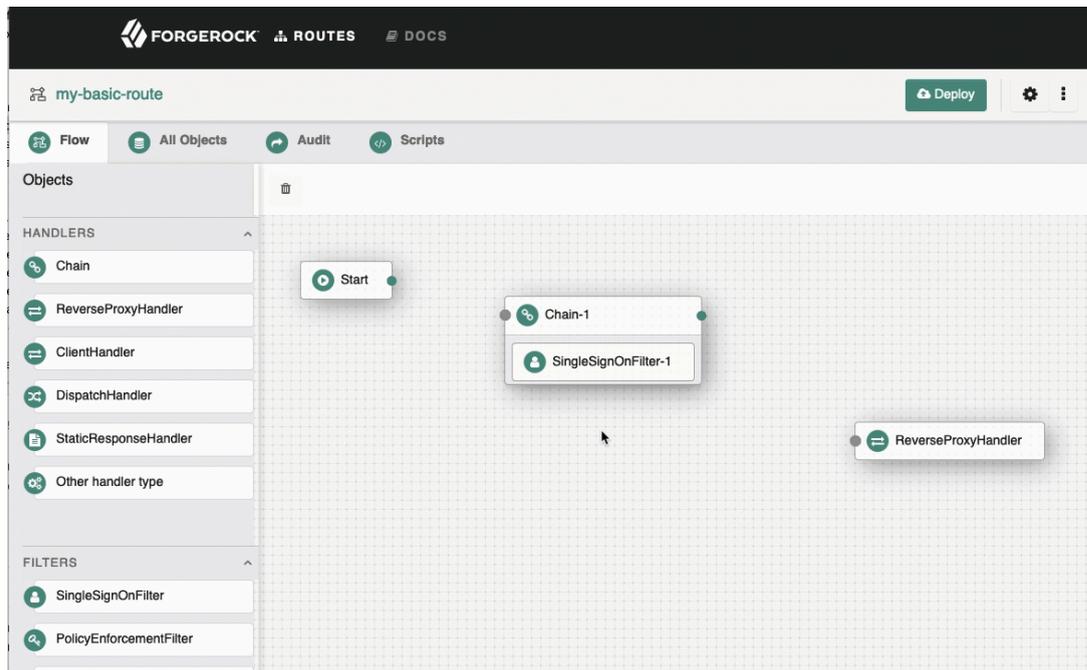
▼ [See how to add objects to the canvas.](#)



3. In the menu for the **SingleSignOnFilter**, enter the name of the **AmService** object you created in [freeform.adoc#studio-add-objects-ff](#), **AmService-1**. The filter uses the object previously defined in the heap.

4. Connect **Start** to **Chain-1**, and **Chain-1** to **ReverseProxyHandler**.

▼ [See how to connect components.](#)



5. On the top-right of the screen, select  and **Display** to review the route.

## Decorating objects in the route

1. Using the route created in [freeform.adoc#studio-create-route-ff](#), select the  **All Objects** tab.

A list of objects in the route is displayed. By default, all available decorators are included in the route heap, but they do not decorate any objects.

2. For the ReverseProxyHandler or filter, select , select the **Decorations** tab, and then enable one or more of the decorators.

3. On the top-right of the screen, select  and **Display** to review the route.

## Editing and importing routes

The following sections describe basic tasks for editing and importing routes in Studio:

### Editing routes in editor mode

After creating a route in Studio, you can edit it by using the options offered in Studio, or by switching to editor mode and using the JSON editor.

Routes created only in the menus of structured editor have the icon . Routes created only in the menus of freeform designer have the icon . Imported routes and routes edited in editor mode have the icon .

## IMPORTANT

When you go into editor mode, you can use the JSON editor to manually edit the route, but can no longer use the full Studio interface to add to or edit the configuration.

1. In Studio, select  **ROUTES**, and then select a route created with the structured editor (with the  icon).
2. Edit the route in Studio or manually:
  - To edit in Studio, select options offered in Studio.
  - To edit manually, select  and **Editor mode**, and use the JSON editor to edit the route.

If the route status is  **Deployed**, it changes to  **Changes pending**.

3. Deploy the route as described in [Deploying and undeploying routes](#).

## Importing routes into Studio

When you import a route into Studio, it is imported in editor mode. You can use the JSON editor to manually edit the route, but can't use the full Studio interface to add or edit filters.

Routes created only in the menus of structured editor have the icon . Routes created only in the menus of freeform designer have the icon . Imported routes and routes edited in editor mode have the icon .

1. In Studio, select  **ROUTES** and then  **Import a route**.
2. Click in the window to import a route, or drag a route from your filesystem.

If the route has a `name` property, the name is automatically used for the **Name** and **ID** fields in Studio.

3. If necessary, make the following changes, and then select **Import**:
  - If the **Name** and **ID** fields are empty, enter a unique name and ID for the route.
  - If the **Name** and **ID** fields are outlined in red, the route name or ID already exists in Studio. Change the name and ID to be unique.
  - If an error message is displayed, the route is not valid JSON. Fix the route and then try again to import it.

The route is added to the list of routes on the  **ROUTES** page.

4. Deploy the route as described in [Deploying and undeploying routes](#).

## Viewing and searching for routes in your configuration

All of the routes that exist in your backend configuration are displayed on the  **ROUTES** page, including imported routes and routes created outside of Studio.

To search for a route, select  **ROUTES**, and type part of the route name in the search box. Matching routes are displayed as you enter the search criteria.

## Restricting access to Studio

When IG is running in development mode, by default the Studio endpoint is open and accessible. To allow only specific users to access Studio, configure a `StudioProtectionFilter` with a `SingleSignOnFilter` or `CrossDomainSingleSignOnFilter`.

The following example uses a `SingleSignOnFilter` to require users to authenticate with AM before they can access Studio, and protects the request from Cross Site Request Forgery (CSRF) attacks.

### 1. Set up AM:

a. (From AM 6.5.3) Select **Services > Add a Service**, and add a **Validation Service** with the following **Valid goto URL Resources**:

- `http://ig.example.com:8080/*`
- `http://ig.example.com:8080/*?*`

b. Select **Applications > Agents > Identity Gateway**, and add an agent with the following values:

- **Agent ID:** `ig_agent`
- **Password:** `password`

For AM 6.5.x and earlier versions, set up an agent as described in [Set up an IG agent in AM 6.5 and earlier](#).

### 2. Set up IG:

a. Set an environment variable for the IG agent password, and then restart IG:

```
$ export AGENT_SECRET_ID='cGFzZ3dvcmQ='
```

The password is retrieved by a `SystemAndEnvSecretStore`, and must be base64-encoded.

b. Add the following route to IG:

Linux

Windows

```
$HOME/.openig/config/admin.json
```

Standalone mode

Web container mode

```
{
  "prefix": "openig",
  "mode": "DEVELOPMENT",
  "properties": {
    "SsoTokenCookieOrHeader": "iPlanetDirectoryPro"
  },
  "connectors": [
    {
      "port": 8080
    },
    {
      "port": 8443
    }
  ],
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider":
          "SystemAndEnvSecretStore-1",
        "url":
          "http://am.example.com:8088/openam/",
        "ssoTokenHeader": "&
          {SsoTokenCookieOrHeader}",
        "version": "7.2"
      }
    }
  ]
}
```

```

    },
    {
      "name": "StudioProtectionFilter",
      "type": "ChainOfFilters",
      "config": {
        "filters": [
          {
            "type": "SingleSignOnFilter",
            "config": {
              "amService": "AmService-1"
            }
          },
          {
            "type": "CsrfFilter",
            "config": {
              "cookieName": "&
{SsoTokenCookieOrHeader}",
              "failureHandler": {
                "type": "StaticResponseHandler",
                "config": {
                  "status": 403,
                  "headers": {
                    "Content-Type": [
                      "text/plain"
                    ]
                  },
                  "entity": "Request forbidden"
                }
              }
            }
          }
        ]
      }
    }
  ]
}

```

Notice the following features of the file:

- The `prefix` sets the base of the administrative route to the default value `/openig`. The Studio endpoint is therefore `/openig/studio`.
- The `mode` is `development`, so by default the Studio endpoint is open and unfiltered.

- The `properties` object sets a configuration parameter for the value of the SSO token cookie or header, which is used in `AmService` and `CorsFilter`.
- The `AmService` uses the IG agent in AM for authentication.

The agent password for `AmService` is provided by a `SystemAndEnvSecretStore` in the heap.

- The `StudioProtectionFilter` calls the `SingleSignOnFilter` to redirect unauthenticated requests to AM, and uses the `CsrfFilter` to protect requests from CSRF attacks. For more information, see [SingleSignOnFilter](#) and [CsrfFilter](#).

c. Restart IG to take into account the changes to `admin.json`.

### 3. Test the setup:

- a. If you are logged in to AM, log out and clear any cookies.
- b. Go to <http://ig.example.com:8080/openig/studio>. The `SingleSignOnFilter` redirects the request to AM for authentication.
- c. Log in to AM with user `demo`, password `Ch4ng31t`. The Studio **Routes** screen is displayed.

## Example routes created with Structured Editor

The following sections give examples of how to set up some of the routes used in the [Gateway guide](#) by using the structured editor of Studio:

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

### Single sign-on in Structured Editor

This section describes how to set up SSO in the structured editor of Studio. For more information about setting up SSO, see [Single sign-on and cross-domain single sign-on](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **≡ Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`

- **Condition: Path:** /home/sso-studio

- **Name:** sso-studio

### 3. Configure authentication:

a. Select  **Authentication**.

b. Select **Single Sign-On**, and enter the following information:

- **AM service** : Configure an AM service to use for authentication:

- **URI:** http://am.example.com:8088/openam

- **Agent** :

- **Username** : ig\_agent

- **Password** : password

Leave all other values as default.

4. On the top-right of the screen, select  and  **Display** to review the route.

```
{
  "name": "sso-studio",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/sso-
studio')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://am.example.com:8088/openam",
        "realm": "/",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
```

```

        "type": "SingleSignOnFilter",
        "config": {
            "amService": "AmService-1"
        }
    },
    ],
    "handler": "ReverseProxyHandler"
}
}
}

```

5. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Policy enforcement in Structured Editor

This section describes how to set up IG as a policy enforcement point in the structured editor of Studio. For more information about setting up policy enforcement, see [Enforce policy decisions from AM](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio> , and select **+ Create a route**.
  - b. Select  **Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`
  - **Condition: Path:** `/home/pep-sso`
  - **Name :** `pep-sso`

The structured editor is displayed.

3. Configure authentication:
  - a. Select  **Authentication**.
  - b. Select **Single Sign-On**, and enter the following information:
    - **AM service :** Configure an AM service to use for authentication:
      - **URI:** `http://am.example.com:8088/openam`
      - **Agent :** The credentials of the agent you created in AM.
        - **Username :** `ig_agent`

- **Password** : password

Leave all other values as default.

4. Configure a PolicyEnforcementFilter:

- a. Select  **Authorization**.
- b. Select **AM Policy Enforcement**, and then select the following options:
  - Access Management configuration:
    - **AM service** : http://am.example.com:8088/openam (/) .
  - Access Management policies:
    - **Policy set** : PEP-SSO
    - **AM SSO token** : \${contexts.ssoToken.value}

Leave all other values as default.

5. On the top-right of the screen, select  and  **Display** to review the route.

```
{
  "name": "pep-sso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/pep-
sso')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
```

```

"filters": [
  {
    "name": "SingleSignOnFilter-1",
    "type": "SingleSignOnFilter",
    "config": {
      "amService": "AmService-1"
    }
  },
  {
    "name": "PolicyEnforcementFilter-1",
    "type": "PolicyEnforcementFilter",
    "config": {
      "application": "PEP-SSO",
      "ssoTokenSubject":
"${contexts.ssoToken.value}",
      "amService": "AmService-1"
    }
  }
],
"handler": "ReverseProxyHandler"
}
}
}

```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Policy enforcement for CDSSO in Structured Editor

This section describes how to set up IG as a policy enforcement point for CDSSO in the structured editor of Studio. For more information about how to set up SSO, see [Enforce AM Policy Decisions In Different Domains](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio> , and select **+ Create a route**.
  - b. Select **≡ Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`

- **Condition: Path:** /home/pep-cdsso

- **Name :** pep-cdsso

### 3. Configure authentication:

a. Select  **Authentication**.

b. Select **Cross-Domain Single Sign-On**, and enter the following information:

- **AM service :**
  - **URI:** http://am.example.com:8088/openam
  - **Version:** The version of the AM instance, for example, 7.2 .
  - **Agent :** The credentials of the agent you created in AM.
    - **Username :** ig\_agent\_cdsso
    - **Password :** password
- **Redirect endpoint :** /home/pep-cdsso/redirect
- **Authentication cookie :**
  - **Path :** /home

Leave all other values as default.

### 4. Configure a PolicyEnforcementFilter:

a. Select  **Authorization**.

b. Select **AM Policy Enforcement**, and select the following options to reflect the configuration of the IG agent in AM:

- Access Management configuration:
  - **AM service :** http://am.example.com:8088/openam (/) .
- Access Management policies:
  - **Policy set :** PEP-CDSSO
  - **AM SSO token ID :** \${contexts.cdsso.token}

Leave all other values as default.

5. On the top-right of the screen, select **:** and  **Display** to review the route.

```
{
  "name": "pep-cdsso",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/pep-cdsso')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    }
  ]
}
```

```

    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "agent": {
          "username": "ig_agent_cdsso",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "url": "http://am.example.com:8088/openam/",
        "version": "7.2"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
            "redirectEndpoint": "/home/pep-
cdsso/redirect",
            "authCookie": {
              "path": "/home",
              "name": "ig-token-cookie"
            },
            "amService": "AmService-1"
          }
        },
        {
          "name": "PolicyEnforcementFilter-1",
          "type": "PolicyEnforcementFilter",
          "config": {
            "application": "PEP-CDSSO",
            "ssoTokenSubject": "${contexts.cdsso.token}",
            "amService": "AmService-1"
          }
        }
      ]
    },
    "handler": "ReverseProxyHandler"
  }
}

```

```
}  
}
```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Token validation using the introspection endpoint in Structured Editor

This section sets up IG as an OAuth 2.0 resource server, using the introspection endpoint, in the structured editor of Studio.

1. Set up AM as described in [Validate access tokens through the introspection endpoint](#). In addition, create an OAuth 2.0 Client authorized to introspect tokens, with the following values:
  - **Client ID**: resource-server
  - Client secret password
  - **Scope(s)**: am-introspect-all-tokens
2. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select  **Structured** to use the structured editor.
  - c. Create a route with the following option:
    - Application URL: `http://app.example.com:8081/rs-introspect-se`
3. Configure authorization:
  - a. Select  **Authorization** > **OAuth 2.0 Resource Server**, and then select the following options:
    - **Token resolver configuration:**
      - **Access token resolver**: OAuth 2.0 introspection endpoint
      - **Introspection endpoint URI**:  
`http://am.example.com:8088/openam/oauth2/introspect`
      - **Client name** and **Client secret**: resource-server and password

This is the name and password of the OAuth 2.0 client with the scope to examine (introspect) tokens, configured in AM.

- Scope configuration:
  - **Evaluate scopes:** Statically
  - **Scopes:** mail, employeenumber
- **OAuth 2.0 Authorization settings:**
  - **Require HTTPS:** Deselect this option
  - **Enable cache:** Deselect this option

Leave all other values as default.

#### 4. Add a StaticResponseHandler:

- a. On the top-right of the screen, select  and  **Editor mode** to switch into editor mode.

##### WARNING

After switching to Editor mode, you cannot go back. You will be able to use the JSON file editor to manually edit the route, but will no longer be able use the full Studio interface to add or edit filters.

- b. Replace the last ReverseProxyHandler in the route with the following StaticResponseHandler, and then save the route:

```
"handler": {
  "type": "StaticResponseHandler",
  "config": {
    "status": 200,
    "headers": {
      "Content-Type": [ "text/html; charset=UTF-8" ]
    },
    "entity": "<html><body><h2>Decoded access_token:
    ${contexts.oauth2.accessToken.info}</h2></body></html>"
  }
}
```

5. On the top-right of the screen, select  and  **Display** to review the route.

```
{
  "name": "rs-introspect-se",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/rs-introspect-
  se')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
```

```

    {
      "name": "OAuth2ResourceServerFilter-1",
      "type": "OAuth2ResourceServerFilter",
      "config": {
        "scopes": [
          "mail",
          "employeenumber"
        ],
        "requireHttps": false,
        "realm": "OpenIG",
        "accessTokenResolver": {
          "name": "token-resolver-1",
          "type":
"TokenIntrospectionAccessTokenResolver",
          "config": {
            "endpoint":
"http://am.example.com:8088/openam/oauth2/introspect",
            "providerHandler": {
              "type": "Chain",
              "config": {
                "filters": [
                  {
                    "type": "HeaderFilter",
                    "config": {
                      "messageType": "request",
                      "add": {
                        "Authorization": [
                          "Basic
${encodeBase64('resource-server:password')}"
                        ]
                      }
                    }
                  }
                ],
                "handler": "ForgeRockClientHandler"
              }
            }
          }
        ],
        "handler": {
          "type": "StaticResponseHandler",
          "config": {

```

```
"status": 200,
"headers": {
  "Content-Type": [ "text/html; charset=UTF-8" ]
},
"entity": "<html><body><h2>Decoded access_token:
${contexts.oauth2.accessToken.info}</h2></body></html>"
}
```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## OpenID Connect in Structured Editor

This section describes how to set up IG as an OpenID Connect relying party in the structured editor of Studio. For more information, see [Use AM as a single OpenID Connect provider](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`
  - **Condition: Path:** `/home/id_token`
  - **Name:** `07-openid`
3. Configure authentication:
  - a. Select **Authentication**.
  - b. Select **OpenID Connect**, and then select the following options:
    - **Client Filter:**
      - **Client Endpoint:** `/home/id_token`
      - **Require HTTPS:** Deselect this option
    - **Client Registration:**

- **Client ID:** oidc\_client
- **Client secret:** password
- **Scopes:** openid, profile, and email
- **Basic authentication:** Select this option
- **Issuer:**
  - **Well-known Endpoint:**  
http://am.example.com:8088/openam/oauth2/.well-known/openid-configuration

Leave all other values as default.

4. On the top-right of the screen, select  and  **Display** to review the route:

```
{
  "name": "07-openid",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path,
'^/home/id_token')}",
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AuthenticatedRegistrationHandler-1",
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name":
"ClientSecretBasicAuthenticationFilter-1",
            "type":
"ClientSecretBasicAuthenticationFilter",
            "config": {
              "clientId": "oidc_client",
              "clientSecretId": "oidc.secret.id",
              "secretsProvider": "SystemAndEnvSecretStore-
1"
            }
          }
        ],
        "handler": "ForgeRockClientHandler"
      }
    }
  ]
}
```

```

],
"handler": {
  "type": "Chain",
  "config": {
    "filters": [
      {
        "name": "AuthorizationCodeOAuth2ClientFilter-1",
        "type": "AuthorizationCodeOAuth2ClientFilter",
        "config": {
          "clientEndpoint": "/home/id_token",
          "failureHandler": {
            "type": "StaticResponseHandler",
            "config": {
              "status": 500,
              "headers": {
                "Content-Type": [
                  "text/plain"
                ]
              },
              "entity": "Error in OAuth 2.0 setup."
            }
          },
          "registrations": [
            {
              "name": "oidc-user-info-client",
              "type": "ClientRegistration",
              "config": {
                "clientId": "oidc_client",
                "issuer": {
                  "name": "Issuer",
                  "type": "Issuer",
                  "config": {
                    "wellKnownEndpoint":
"http://am.example.com:8088/openam/oauth2/.well-known/openid-configuration"
                  }
                },
                "scopes": [
                  "openid",
                  "profile",
                  "email"
                ],
                "authenticatedRegistrationHandler":
"AuthenticatedRegistrationHandler-1"
              }
            }
          ]
        }
      }
    ]
  }
}

```

```
        }
      ],
      "requireHttps": false,
      "cacheExpiration": "disabled"
    }
  }
],
"handler": "ReverseProxyHandler"
}
}
}
```

5. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Token transformation in Structured Editor

This section describes how to set up token transformation in the structured editor of Studio. For more information about setting up token transformation, see [Transform OpenID Connect ID tokens into SAML assertions](#).

1. In IG Studio, create a route:
  - a. Go to `http://ig.example.com:8080/openig/studio` , and select **+ Create a route**.
  - b. Select  **Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`
  - **Condition:** Path: `/home/id_token`
  - **Name:** `50-idtoken`
3. Configure authentication:
  - a. Select  **Authentication**.
  - b. Select **OpenID Connect**, and enter the following information:
    - **Client Filter :**
      - **Client Endpoint:** `/home/id_token`
      - **Require HTTPS:** Deselect this option
    - **Client Registration :**

- **Client ID** : oidc\_client
- **Client secret** : password
- **Scopes**: openid, profile, and email
- **Basic authentication**: Select this option
- **Issuer** :
  - Well-known endpoint:  
http://am.example.com:8088/openam/oauth2/.well-known/openid-configuration

Leave all other values as default, and save your settings.

4. Set up token transformation:

- a. Select and enable **Token transformation**.
- b. Enter the following information:
  - **AM service** : Configure an AM service to use for authentication and REST STS requests.
    - **URI**: http://am.example.com:8088/openam
    - **Agent** : The credentials of the agent you created in AM.
      - **Username** : ig\_agent
      - **Password** : password
  - **Username** : oidc\_client
  - **Password** : password
  - **id\_token** : \${attributes.openid.id\_token}
  - **Instance** : openidg

5. Add a StaticResponseHandler:

- a. On the top-right of the screen, select  and  **Editor mode** to switch into editor mode.

**WARNING**

After switching to Editor mode, you cannot go back. You will be able to use the JSON file editor to manually edit the route, but will no longer be able use the full Studio interface to add or edit filters.

- b. Replace the last ReverseProxyHandler in the route with the following StaticResponseHandler, and then save the route:

```
"handler" : {
  "type" : "StaticResponseHandler",
  "config" : {
```



```

    "name": "AmService-1",
    "type": "AmService",
    "config": {
      "agent": {
        "username": "ig_agent",
        "passwordSecretId": "agent.secret.id"
      },
      "secretsProvider": "SystemAndEnvSecretStore-1",
      "url": "http://am.example.com:8088/openam/",
      "version": "7.2"
    }
  },
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "AuthorizationCodeOAuth2ClientFilter-1",
          "type": "AuthorizationCodeOAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/id_token",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/plain"
                  ]
                },
              },
              "entity": "An error occurred during the
OAuth2 setup."
            }
          },
        },
      ],
      "registrations": [
        {
          "name": "oidc-user-info-client",
          "type": "ClientRegistration",
          "config": {
            "clientId": "oidc_client",
            "issuer": {
              "name": "Issuer",
              "type": "Issuer",
              "config": {

```

```

        "wellKnownEndpoint":
"http://am.example.com:8088/openam/oauth2/.well-
known/openid-configuration"
    }
  },
  "scopes": [
    "openid",
    "profile",
    "email"
  ],
  "authenticatedRegistrationHandler":
"AuthenticatedRegistrationHandler-1"
}
}
],
"requireHttps": false,
"cacheExpiration": "disabled"
}
},
{
  "name": "TokenTransformationFilter-1",
  "type": "TokenTransformationFilter",
  "config": {
    "idToken": "${attributes.openid.id_token}",
    "instance": "openid",
    "amService": "AmService-1"
  }
}
],
"handler": {
  "type": "StaticResponseHandler",
  "config": {
    "status": 200,
    "headers": {
      "Content-Type": [ "text/plain; charset=UTF-8"
]
    }
  },
  "entity": "
{\"id_token\":\n\"${attributes.openid.id_token}\n\"
\n\n\n{\"saml_assertions\":\n\n\"${contexts.sts.issuedToken}
\n}\""
}
}
}
}

```

```
}  
}
```

7. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Simple throttling filter in Structured Editor

This section describes how to set up a simple throttling filter in the structured editor of Studio. For more information about how to set up throttling, see [Configure Simple Throttling](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **☰ Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`
  - **Condition: Path:** `/home/throttle-simple`
  - **Name:** `00-throttle-simple`
3. Select and enable **▼ Throttling**.
4. In **GROUPING POLICY**, apply the rate to a single group.

All requests are grouped together, and the default throttling rate is applied to the group. By default, no more than 100 requests can access the sample application each second.

5. In **RATE POLICY**, select **Fixed**, and allow 6 requests each 10 seconds.
6. On the top-right of the screen, select **ⓘ** and **🖥 Display** to review the route.

```
{  
  "name": "00-throttle-simple",  
  "baseURI": "http://app.example.com:8081",  
  "condition": "${find(request.uri.path, '^/home/throttle-simple')}",  
  "handler": {  
    "type": "Chain",  
    "config": {  
      "filters": [  

```

```

    {
      "type": "ThrottlingFilter",
      "name": "ThrottlingFilter-1",
      "config": {
        "requestGroupingPolicy": "",
        "rate": {
          "numberOfRequests": 6,
          "duration": "10 s"
        }
      }
    },
    ],
    "handler": "ReverseProxyHandler"
  }
}
}
}

```

7. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Mapped throttling filter in Structured Editor

This section describes how to set up a mapped throttling filter in the structured editor of Studio. For more information about how to set up throttling, see [Configure Mapped Throttling](#).

1. Set up AM as described in [Validate access tokens through the introspection endpoint](#). In addition, create an OAuth 2.0 Client authorized to introspect tokens, with the following values:
  - **Client ID**: resource-server
  - Client secret password
  - **Scope(s)**: am-introspect-all-tokens
2. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio> , and select **+ Create a route**.
  - b. Select  **Structured** to use the structured editor.
3. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI**: `http://app.example.com:8081`

- **Condition: Path:** /home/throttle-mapped-se

- **Name:** 00-throttle-mapped-se

#### 4. Configure authorization:

a. Select **Authorization > OAuth 2.0 Resource Server**, and then select the following options:

- **Token resolver configuration:**

- **Access token resolver:** OAuth 2.0 introspection endpoint

- **Introspection endpoint URI:**

- http://am.example.com:8088/openam/oauth2/introspect

- **Client name and Client secret:** resource-server and password

This is the name and password of the OAuth 2.0 client with the scope to examine (introspect) tokens, configured in AM.

- **Scope configuration:**

- **Evaluate scopes:** Statically

- **Scopes:** mail, employeenumber

- **OAuth 2.0 Authorization settings:**

- **Require HTTPS:** Deselect this option

- **Enable cache:** Deselect this option

Leave all other values as default.

#### 5. Configure throttling:

a. Select and enable **Throttling**.

b. Set up the grouping policy:

i. In **GROUPING POLICY**, apply the rate to independent groups of requests.

Requests are split into different groups according to criteria, and the throttling rate is applied to each group.

ii. Select to group requests by custom criteria.

Enter `${contexts.oauth2.accessToken.info.mail}` as the custom expression. This expression defines the subject in the OAuth2Context.

c. Set up the rate policy:

i. In **RATE POLICY**, select **Mapped**.

ii. Select to map requests by custom criteria.

- iii. Enter the custom expression  
`${contexts.oauth2.accessToken.info.status}`.
  - iv. In **Default Rate**, select **Edit** and change default rate to 1 request each 10 seconds.
  - v. In **Mapped Rates**, add the following rate for `gold` status:
    - **Match Value** : `gold`
    - **Number of requests** : 6
    - **Period** : 10 seconds
  - vi. Add a different rate for `silver` status:
    - **Match Value** : `silver`
    - **Number of requests** : 3
    - **Period** : 10 seconds
  - vii. Add a different rate for `bronze` status:
    - **Match Value** : `bronze`
    - **Number of requests** : 1
    - **Period** : 10 seconds
  - viii. Save the rate policy.
6. Select  **Chain**, and change the order of the filters so that  **Throttling** comes after  **Authorization**.
  7. On the top-right of the screen, select  and  **Display** to review the route.

```
{
  "name": "00-throttle-mapped-se",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-mapped-se')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "0Auth2ResourceServerFilter-1",
          "type": "0Auth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,

```

```

    "realm": "OpenIG",
    "accessTokenResolver": {
      "name": "token-resolver-1",
      "type":
"TokenIntrospectionAccessTokenResolver",
      "config": {
        "endpoint":
"http://am.example.com:8088/openam/oauth2/introspect",
        "providerHandler": {
          "type": "Chain",
          "config": {
            "filters": [
              {
                "type": "HeaderFilter",
                "config": {
                  "messageType": "request",
                  "add": {
                    "Authorization": [
                      "Basic
${encodeBase64('resource-server:password')}"
                    ]
                  }
                }
              }
            ],
            "handler": "ForgeRockClientHandler"
          }
        }
      }
    },
    {
      "name": "ThrottlingFilter-1",
      "type": "ThrottlingFilter",
      "config": {
        "requestGroupingPolicy":
"${contexts.oauth2.accessToken.info.mail}",
        "throttlingRatePolicy": {
          "name": "MappedPolicy",
          "type": "MappedThrottlingPolicy",
          "config": {
            "throttlingRateMapper":
"${contexts.oauth2.accessToken.info.status}",
            "throttlingRatesMapping": {

```

```

    "gold": {
      "numberOfRequests": 6,
      "duration": "10 s"
    },
    "silver": {
      "numberOfRequests": 3,
      "duration": "10 s"
    },
    "bronze": {
      "numberOfRequests": 1,
      "duration": "10 s"
    }
  },
  "defaultRate": {
    "numberOfRequests": 1,
    "duration": "10 s"
  }
}
}
}
}
}
],
"handler": "ReverseProxyHandler"
}
}
}
}
}

```

8. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Scriptable throttling filter in Structured Editor

This section describes how to set up a scriptable throttling filter in the structured editor of Studio. For more information about how to set up throttling, see [Configure scriptable throttling](#).

1. Set up AM as described in [Validate access tokens through the introspection endpoint](#). In addition, create an OAuth 2.0 Client authorized to introspect tokens, with the following values:
  - **Client ID:** resource-server
  - **Client secret:** password

- **Scope(s):** am-introspect-all-tokens

2. In IG Studio, create a route:

- Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
- Select **☰ Structured** to use the structured editor.

3. Select **Advanced options** on the right, and create a route with the following options:

- **Base URI:** `http://app.example.com:8081`
- **Condition: Path:** `/home/throttle-scriptable-se`
- **Name:** `00-throttle-scriptable-se`

4. Configure authorization:

- Select **🔑 Authorization > OAuth 2.0 Resource Server**, and then select the following options:

- **Token resolver configuration:**

- **Access token resolver:** OAuth 2.0 introspection endpoint
- **Introspection endpoint URI:**  
`http://am.example.com:8088/openam/oauth2/introspect`
- **Client name and Client secret:** resource-server and password

This is the name and password of the OAuth 2.0 client with the scope to examine (introspect) tokens, configured in AM.

- **Scope configuration:**

- **Evaluate scopes:** Statically
- **Scopes:** mail, employeenumber

- **OAuth 2.0 Authorization settings:**

- **Require HTTPS:** Deselect this option
- **Enable cache:** Deselect this option

Leave all other values as default.

5. Configure throttling:

- Select and enable **▼ Throttling**.
- Set up the grouping policy:
  - In **GROUPING POLICY**, apply the rate to independent groups of requests.

Requests are split into different groups according to criteria, and the throttling rate is applied to each group.

- ii. Select to group requests by custom criteria.
  - iii. Enter `${contexts.oauth2.accessToken.info.mail}` as the custom expression.
- c. Set up the rate policy:
- i. In **RATE POLICY**, select **Scripted**.
  - ii. Select to create a new script, and name it `X-User-Status`. So that you can easily identify the script, use a name that describes the content of the script.
  - iii. Add the following argument/value pairs:
    - **argument:** `status`, **value:** `gold`
    - **argument:** `rate`, **value:** `6`
    - **argument:** `duration`, **value:** `10 seconds`
      - Replace the default script with the content of a valid Groovy script. For example, enter the following script:

```
if (contexts.oauth2.accessToken.info.status
== status) {
    return new ThrottlingRate(rate, duration)
} else {
    return null
}
```

Alternatively, skip the step to define arguments, and add the following script instead:

```
if (contexts.oauth2.accessToken.info.status
== 'gold') {
    return new ThrottlingRate(6, '10 seconds')
} else {
    return null
}
```

**NOTE**

Studio does not check the validity of the Groovy script.

- iv. Enable the default rate, and set it to 1 request each 10 seconds.
  - v. Save the rate policy. The script is added to the list of reference scripts available to use in scriptable throttling filters.
6. Select  **Chain**, and change the order of the filters so that  **Throttling** comes after  **Authorization**.

7. On the top-right of the screen, select **:** and  **Display** to review the route.

```
{
  "name": "00-throttle-scriptable-se",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/throttle-scriptable-se')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "OAuth2ResourceServerFilter-1",
          "type": "OAuth2ResourceServerFilter",
          "config": {
            "scopes": [
              "mail",
              "employeenumber"
            ],
            "requireHttps": false,
            "realm": "OpenIG",
            "accessTokenResolver": {
              "name": "token-resolver-1",
              "type":
"TokenIntrospectionAccessTokenResolver",
              "config": {
                "endpoint":
"http://am.example.com:8088/openam/oauth2/introspect",
                "providerHandler": {
                  "type": "Chain",
                  "config": {
                    "filters": [
                      {
                        "type": "HeaderFilter",
                        "config": {
                          "messageType": "request",
                          "add": {
                            "Authorization": [
                              "Basic
${encodeBase64('resource-server:password')}"
                            ]
                          }
                        }
                      ]
                    }
                  }
                }
              ]
            },
            "type": "Chain",
            "config": {
              "filters": [
                {
                  "type": "HeaderFilter",
                  "config": {
                    "messageType": "request",
                    "add": {
                      "Authorization": [
                        "Basic
${encodeBase64('resource-server:password')}"
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      ]
    }
  }
}
```



```
    ],  
    "handler": "ReverseProxyHandler"  
  }  
}  
}
```

8. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Proxy for websocket traffic in Structured Editor

This section describes how to set up IG to proxy WebSocket traffic, in the structured editor of Studio. For more information about how to set up proxying for WebSocket traffic, see [Proxy WebSocket traffic](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select  **Structured** to use the structured editor.
2. Select **Advanced options** on the right, and create a route with the following options:
  - o **Base URI:** `http://app.example.com:8081`
  - o **Condition: Path:** `/websocket-se`
  - o **Name:** `websocket-se`
  - o **Enable WebSocket:** Select this option
3. Configure authentication:
  - a. Select  **Authentication**.
  - b. Select **Single Sign-On**, and enter the following information:
    - **URI:** `http://am.example.com:8088/openam`
    - **Version:** The version of the AM instance, for example, `7.2`.
    - **Agent:** The credentials of the Java agent you created in AM.
      - **Username:** `ig_agent`
      - **Password:** `password`

Leave all other values as default.
4. On the top-right of the screen, select  and **Display** to review the route.

```

{
  "name": "websocket-se",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/websocket-se')}",
  "heap": [
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "http://am.example.com:8088/openam",
        "realm": "/",
        "version": "7.2",
        "agent": {
          "username": "ig_agent",
          "password": "password"
        },
        "sessionCache": {
          "enabled": false
        }
      }
    },
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "config": {
        "websocket": {
          "enabled": true
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SingleSignOnFilter-1",
          "type": "SingleSignOnFilter",
          "config": {
            "amService": "AmService-1"
          }
        }
      ]
    },
    "handler": "ReverseProxyHandler"
  }
}

```

```
}  
}  
}
```

5. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Example routes created with Freeform Designer

The following sections give examples of how to use the templates provided by the freeform designer:

In this release, routes generated in Studio do not use the Commons Secrets Service. Documentation examples generated with Studio use deprecated properties.

### Using a basic template in FreeForm Designer

This section describes how to use a basic template in freeform designer to set up SSO. For more information about setting up and testing SSO, see [Single sign-on and cross-domain single sign-on](#).

#### *Use a Basic Template in FreeForm Designer*

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select  **Freeform** to use the freeform designer.
2. Select  **Basic** to create a route from a blank template.
3. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI:** `http://app.example.com:8081`
  - **Condition: Path:** `/home/sso-ff`
  - **Name:** `sso-ff`

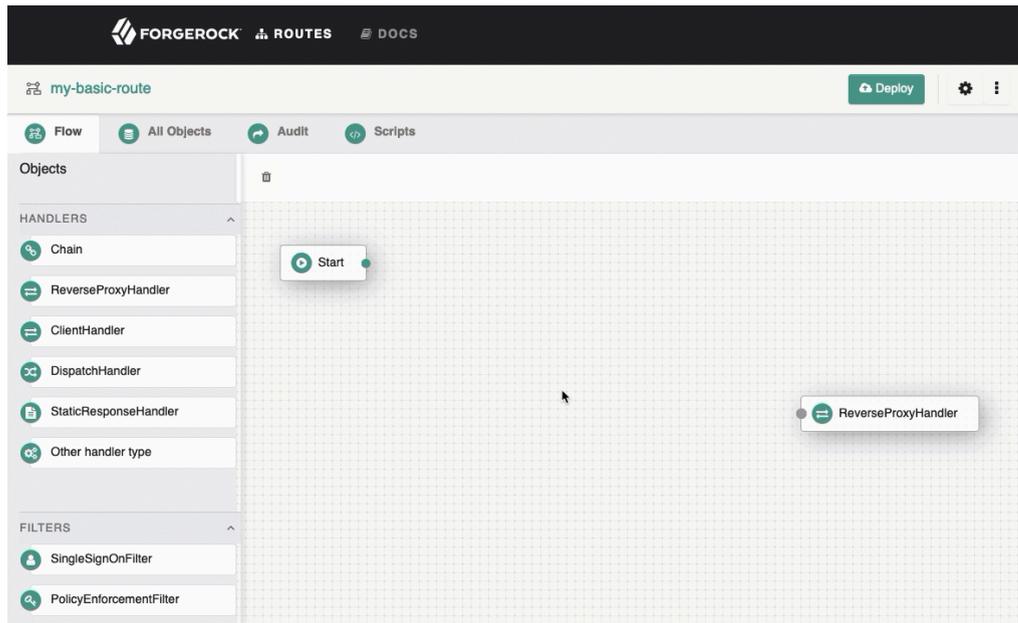
The route is displayed on the  **Flow** tab of the canvas. Select the  **All Objects** tab to view a list of objects in the route.

Double-click on any object to review or edit it. After double-clicking on an object, select the **Decorations** tab to decorate it.

#### 4. Configure authentication with a SingleSignOnFilter:

- a. Select the **Flow** tab, and delete the connector between **Start** and **ReverseProxyHandler**.
- b. From the side bar, drag a **Chain** onto the canvas, and then drag a **SingleSignOnFilter** into the chain.

▼ [See how to add objects to the canvas.](#)

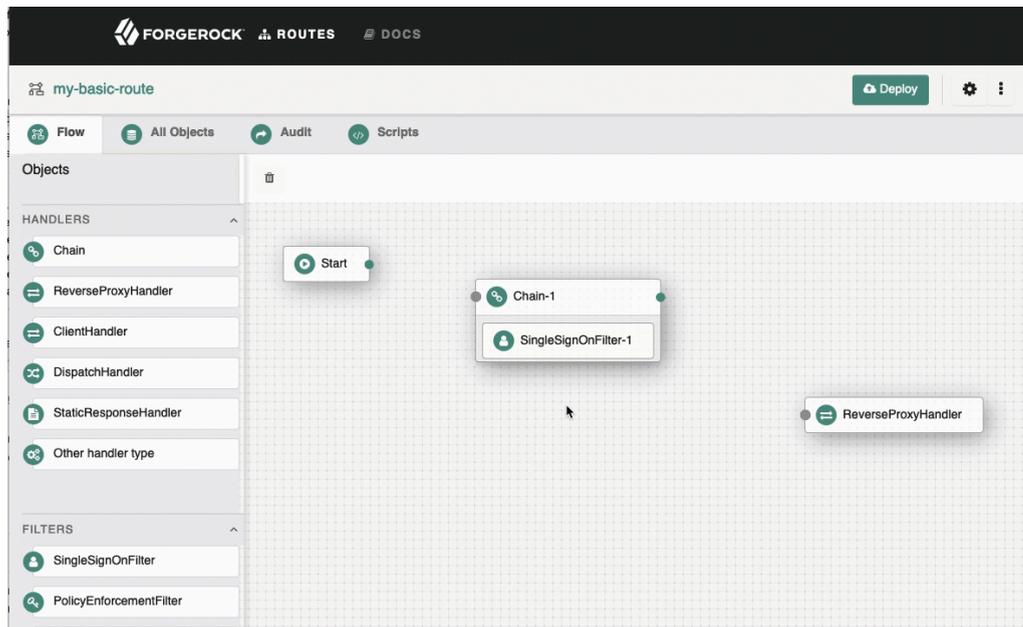


- c. In the **Edit SingleSignOnFilter** page, click **+**, and create an AM service, with the following values:

- **URI:** `http://am.example.com:8088/openam`
- **Agent:**
  - **Username:** `ig_agent`
  - **Password:** `password`

- d. Connect **Start** to **Chain-1**, and **Chain-1** to **ReverseProxyHandler**.

▼ [See how to connect components.](#)



5. On the top-right of the screen, select **:** and **Display** to review the route.

```
{
  "name": "sso-ff",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/sso-ff')}",
  "handler": "Chain-1",
  "heap": [
    {
      "name": "ReverseProxyHandler",
      "type": "ReverseProxyHandler"
    },
    {
      "type": "BaseUriDecorator",
      "name": "baseUri"
    },
    {
      "type": "TimerDecorator",
      "name": "timer",
      "config": {
        "timeUnit": "ms"
      }
    },
    {
      "type": "CaptureDecorator",
      "name": "capture",
      "config": {
        "captureEntity": false,
        "captureContext": false,
        "maxEntityLength": 524288
      }
    }
  ]
}
```

```

    }
  },
  {
    "name": "Chain-1",
    "type": "Chain",
    "config": {
      "handler": "ReverseProxyHandler",
      "filters": [
        "SingleSignOnFilter-1"
      ]
    }
  },
  {
    "name": "AmService-1",
    "type": "AmService",
    "config": {
      "url": "http://am.example.com:8088/openam",
      "realm": "/",
      "agent": {
        "username": "ig_agent",
        "password": "password"
      },
      "sessionCache": {
        "enabled": false
      }
    }
  },
  {
    "name": "SingleSignOnFilter-1",
    "type": "SingleSignOnFilter",
    "config": {
      "amService": "AmService-1"
    }
  }
]
}

```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

## Protecting a web app with Freeform Designer

This section describes how to use freeform designer to protect a web app, using AM for single sign-on and policy enforcement.

The generated route contains a chain of objects to authenticate the user, enforce an AM authorization policy, retrieve the user's profile, insert it into the request, and, finally, forward the request to the web app.

Before you start, set up AM as described in [Enforce policy decisions from AM](#).

1. In IG Studio, create a route:
  - a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **Freeform** to use the freeform designer.
2. Select **Web SSO** to use the template for protecting web apps.
3. Select **Advanced options** on the right, and create a route with the following options:
  - **Base URI** : `http://app.example.com:8081`
  - **Condition: Path** : `/home/pep-sso-ff`
  - **Name** : `pep-sso-ff`
  - **AM Configuration** :
    - **URI** : `http://am.example.com:8088/openam`
    - **Username** : `ig_agent`
    - **Password** : `password`

The route is displayed on the **Flow** tab of the canvas. Select the **All Objects** tab to view a list of objects in the route.

Double-click on any object to review or edit it. After double-clicking on an object, select the **Decorations** tab to decorate it.

4. On the **Flow** tab, double-click the Policy Enforcement object, and add a policy set with the following values:
  - **Policy set** : `PEP-SSO`
  - **AM SSO token** : `${contexts.ssoToken.value}`

Leave all other values as default.

5. On the top-right of the screen, select **Display** to review the route.

```
{
  "name" : "pep-sso-ff",
  "baseURI" : "http://app.example.com:8081",
```

```
"condition": "${find(request.uri.path, '^/home/pep-sso-ff')}",
"handler": "Chain",
"heap": [
  {
    "name": "Chain",
    "type": "Chain",
    "config": {
      "handler": "ReverseProxyHandler",
      "filters": [
        "SSO",
        "PolicyEnforcement",
        "GetEmail",
        "InjectEmail"
      ]
    }
  },
  {
    "name": "SSO",
    "type": "SingleSignOnFilter",
    "config": {
      "amService": "AmService"
    }
  },
  {
    "name": "ReverseProxyHandler",
    "type": "ReverseProxyHandler"
  },
  {
    "name": "AmService",
    "type": "AmService",
    "config": {
      "url": "http://am.example.com:8088/openam",
      "realm": "/",
      "agent": {
        "username": "ig_agent",
        "password": "password"
      },
      "sessionCache": {
        "enabled": false
      }
    }
  },
  {
    "name": "PolicyEnforcement",
```

```

"type": "PolicyEnforcementFilter",
"config": {
  "amService": "AmService",
  "ssoTokenSubject": "${contexts.ssoToken.value}",
  "cache": {
    "enabled": false
  },
  "application": "PEP-SSO"
}
},
{
"name": "GetEmail",
"type": "UserProfileFilter",
"config": {
  "username": "${contexts.ssoToken.info.uid}",
  "userService": {
    "type": "UserProfileService",
    "config": {
      "amService": "AmService"
    }
  }
}
},
{
"name": "InjectEmail",
"type": "HeaderFilter",
"config": {
  "messageType": "REQUEST",
  "add": {
    "Email": [
      "${contexts.userProfile.username}"
    ]
  }
}
},
{
"type": "BaseUriDecorator",
"name": "baseUri"
},
{
"type": "TimerDecorator",
"name": "timer",
"config": {
  "timeUnit": "ms"
}
}

```

```
    },
    {
      "type": "CaptureDecorator",
      "name": "capture",
      "config": {
        "captureEntity": false,
        "captureContext": false,
        "maxEntityLength": 524288
      }
    }
  ]
}
```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

### *Test the Setup*

1. If you are logged in to AM, log out and clear any cookies.
2. Go to <http://ig.example.com:8080/home/pep-sso-ff>.
3. Log in to AM as user `demo`, password `Ch4ng31t`.

AM returns a policy decision that grants access to the sample application.

## Protect an API with Freeform Designer

This section describes how to use freeform designer to protect APIs, using AM as an OAuth 2.0 authorization server.

The generated route contains a chain of objects to authenticate the user, throttle the rate of requests to the API, and, finally, forward the request to the sample app.

Before you start, set up AM as described in [Validate access tokens through the introspection endpoint](#). In addition, create an OAuth 2.0 Client authorized to introspect tokens, with the following values:

- **Client ID**: resource-server
- **Client secret**: password
- **Scope(s)**: am-introspect-all-tokens

1. In IG Studio, create a route:

- a. Go to <http://ig.example.com:8080/openig/studio>, and select **+ Create a route**.
  - b. Select **Freeform** to use the freeform designer.
2. Select **Applications > Agents > Identity Gateway**, add an agent with the following values:
    - **Agent ID** : ig\_agent
    - **Password** : password
    - **Token Introspection** : Realm Only
  3. Select **Advanced options** on the right, and create a route with the following options:
    - **Base URI** : `http://app.example.com:8081`
    - **Condition: Path** : `/home/rs-introspect-ff`
    - **Name** : rs-introspect-ff
    - **AM Configuration** :
      - **URI** : `http://am.example.com:8088/openam`
      - **Username** : ig\_agent
      - **Password** : password
      - **Scopes** : mail, employeenumber

The route is displayed on the **Flow** tab of the canvas.

Notice that the **Start**, **Chain**, and **ReverseProxyHandler** objects are connected by solid lines, but other objects, such as **Authenticate to Am Chain**, are connected by a fading line. Objects connected by a fading line are used by other objects in the route.

Select the **All Objects** tab to view a list of objects in the route. Double-click on any object to review or edit it. After double-clicking on an object, select the **Decorations** tab to decorate it.

4. On the **Flow** tab, double-click the OAuth2RS object, and edit it as follows:
  - **Require HTTPS** : Deselect this option
  - **Realm** : OpenIG

Leave the other values as they are.

5. On the top-right of the screen, select **Display** to review the route.

```
{
  "name": "rs-introspect-ff",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home/rs-
```

```

introspect-ff'}}",
  "handler": "Chain",
  "properties": {
    "amUsername": "ig_agent",
    "amPassword": "password"
  },
  "heap": [
    {
      "name": "ClientHandler",
      "type": "ClientHandler"
    },
    {
      "name": "Chain",
      "type": "Chain",
      "config": {
        "handler": "ReverseProxyHandler",
        "filters": [
          "OAuth2RS",
          "Throttling"
        ]
      }
    },
    {
      "type": "OAuth2ResourceServerFilter",
      "name": "OAuth2RS",
      "config": {
        "requireHttps": false,
        "realm": "OpenIG",
        "scopes": [
          "mail",
          "employeenumber"
        ],
        "accessTokenResolver":
"TokenIntrospectionAccessTokenResolver"
      }
    },
    {
      "type": "TokenIntrospectionAccessTokenResolver",
      "name": "TokenIntrospectionAccessTokenResolver",
      "config": {
        "amService": "AmService",
        "providerHandler": "Authenticate to AM Chain"
      }
    }
  ],
  {

```

```

    "name": "ReverseProxyHandler",
    "type": "ReverseProxyHandler"
  },
  {
    "name": "AmService",
    "type": "AmService",
    "config": {
      "url": "http://am.example.com:8088/openam",
      "realm": "/",
      "agent": {
        "username": "${amUsername}",
        "password": "${amPassword}"
      },
      "sessionCache": {
        "enabled": false
      }
    }
  },
  {
    "name": "Authenticate to AM Chain",
    "type": "Chain",
    "config": {
      "handler": "ClientHandler",
      "filters": [
        "Authenticate to AM Filter"
      ]
    }
  },
  {
    "name": "Authenticate to AM Filter",
    "type": "HeaderFilter",
    "config": {
      "messageType": "REQUEST",
      "add": {
        "Authorization": [
          "Basic ${encodeBase64(join(array(amUsername,
amPassword), ':'))}"
        ]
      }
    }
  },
  {
    "name": "Throttling",
    "type": "ThrottlingFilter",
    "config": {

```

```

        "requestGroupPolicy":
"${contexts.oauth2.info.sub}",
        "rate": {
            "numberOfRequests": 60,
            "duration": "60 s"
        }
    },
    {
        "type": "BaseUriDecorator",
        "name": "baseUri"
    },
    {
        "type": "TimerDecorator",
        "name": "timer",
        "config": {
            "timeUnit": "ms"
        }
    },
    {
        "type": "CaptureDecorator",
        "name": "capture",
        "config": {
            "captureEntity": false,
            "captureContext": false,
            "maxEntityLength": 524288
        }
    }
]
}

```

6. Select  **Deploy** to push the route to the IG configuration.

You can check the `$HOME/.openig/config/routes` folder to see that the route is there.

7. Test the setup:

a. In a terminal window, use a `curl` command similar to the following to retrieve an access token:

```

$ mytoken=$(curl -s \
--user "client-application:password" \
--data
"grant_type=password&username=demo&password=Ch4ng31t&sc
ope=mail%20employeenumber" \

```

```
http://am.example.com:8088/openam/oauth2/access_token |
jq -r ".access_token")
```

b. Validate the access token returned in the previous step:

```
$ curl -v http://ig.example.com:8080/home/rs-
introspect-ff --header "Authorization: Bearer
${mytoken}"
```

The HTML of the sample application is returned.

## Appendix A: Summary of tasks, route status, and icons

The following tables summarize the basic tasks that you can do in Studio, and summarizes the icons and status displayed in Studio:

### Task reference

To do this	Do this
Create a new route	Select  <b>ROUTES</b> , <b>+</b> <b>Create a route</b> .
Select a route	Select  <b>ROUTES</b> , and then select a route to view.
Display the config of a selected route	Select a route, and then select  and  <b>Display</b> .
Deploy a selected route	Select a route, and then select  <b>Deploy</b> .
Undeploy a selected route	Select a deployed route, and then select  and  <b>Undeploy</b> .
Change the basic config of a route	Select a route, and then select  <b>Route settings</b> . Edit the route and save the changes.

### Route status

Status	Description	Action
 <b>Undeployed</b>	The route is saved in Studio but is not deployed to the backend.	Deploy the route. The status changes to  <b>Deployed</b> .

Status	Description	Action
✔ <b>Deployed</b>	The route is saved in Studio and deployed to the backend.	None. The route has the same configuration in Studio and the backend.
! <b>Changes pending</b>	The route has been deployed and then subsequently changed in Studio.	Deploy the route. The status changes to ✔ <b>Deployed</b> .
⚠ <b>Out of sync</b>	The route has been deployed and then subsequently changed in the backend, or in both Studio and the backend.	<p>Select ☁ <b>Deploy</b>. A message informs you that a different version of the route is deployed in the backend. Select an option:</p> <ul style="list-style-type: none"> <li>☁ <b>Deploy</b>: The version in Studio overwrites the backend.</li> <li>⬆ <b>Import a route</b>: The version in the backend overwrites Studio.</li> </ul> <p>When you import a route into Studio you go into <b>editor mode</b>. You can use the JSON editor to manually edit the route, but can no longer use the full Studio interface to add or edit filters.</p>
⚠ <b>Compatibility update required</b>	The route was created in Studio in an earlier version of IG. Some information is needed to complete the upgrade.	Enter the information as prompted, and then select ☁ <b>Deploy</b> to deploy the route.

### Icons

Icon	Mode	Description
------	------	-------------

Icon	Mode	Description
☰	Structured editor	The route was created and edited using the menus and options of structured editor.
{ }	Editor mode	The route was imported into Studio, or was created in Studio and then edited in editor mode.
🛠️	Freeform designer	The route was created on the canvas of freeform designer.

Was this helpful?  

Copyright © 2010-2024 ForgeRock, all rights reserved.