Red Hat Single Sign-On 7.0 Server Developer Guide

Server Developer Guide

Red Hat Customer Content Services

Server Developer Guide

Legal Notice

Abstract

This guide consist of information for developers to customize Red Hat Single Sign-On 7.0

Table of Contents

CHAPTER 1. PREFACE **3**

CHAPTER 2. ADMIN REST API **4**

2.1. EXAMPLE USING CURL 4

CHAPTER 3. THEMES **5**

3.1. THEME TYPES 5

3.2. CONFIGURE THEME 5

3.3. DEFAULT THEMES 6

3.4. CREATING A THEME 6

3.5. DEPLOYING THEMES 10

CHAPTER 4. CUSTOM USER ATTRIBUTES **13**

4.1. REGISTRATION PAGE 13

4.2. ACCOUNT MANAGEMENT CONSOLE 13

CHAPTER 1. PREFACE

In some of the example listings, what is meant to be displayed on one line does not fit inside the available page width. These lines have been broken up. A '\' at the end of a line means that a break has been introduced to fit in the page, with the following lines indented. So:

```
Let's pretend to have an extremely \
long line that \
does not fit
This one is short
```

Is really:

```
Let's pretend to have an extremely long line that does not fit
This one is short
```

CHAPTER 2. ADMIN REST API

Red Hat Single Sign-On comes with a fully functional Admin REST API with all features provided by the Admin Console.

To invoke the API you need to obtain an access token with the appropriate permissions. The required permissions are described in Server Administration Guide.

A token can be obtained by enabling authenticating to your application with Red Hat Single Sign-On, see the Securing Applications and Services Guide. You can also use direct access grant to obtain an access token.

Refer to API Documentation for complete documentation.

2.1. EXAMPLE USING CURL

Obtain access token for user in the realm **`master`** with username **`admin`** and password **`password`**:

```
curl \
  -d "client_id=admin-cli" \
  -d "username=admin" \
  -d "password=password" \
  -d "grant_type=password" \
  "http://localhost:8080/auth/realms/master/protocol/openid-
connect/token"
```

Note

By default this token expires in 1 minute

The result will be a JSON document. To invoke the API you need to extract the value of the **`access_token`** property. You can then invoke the API by including the value in the **`Authorization`** header of requests to the API.

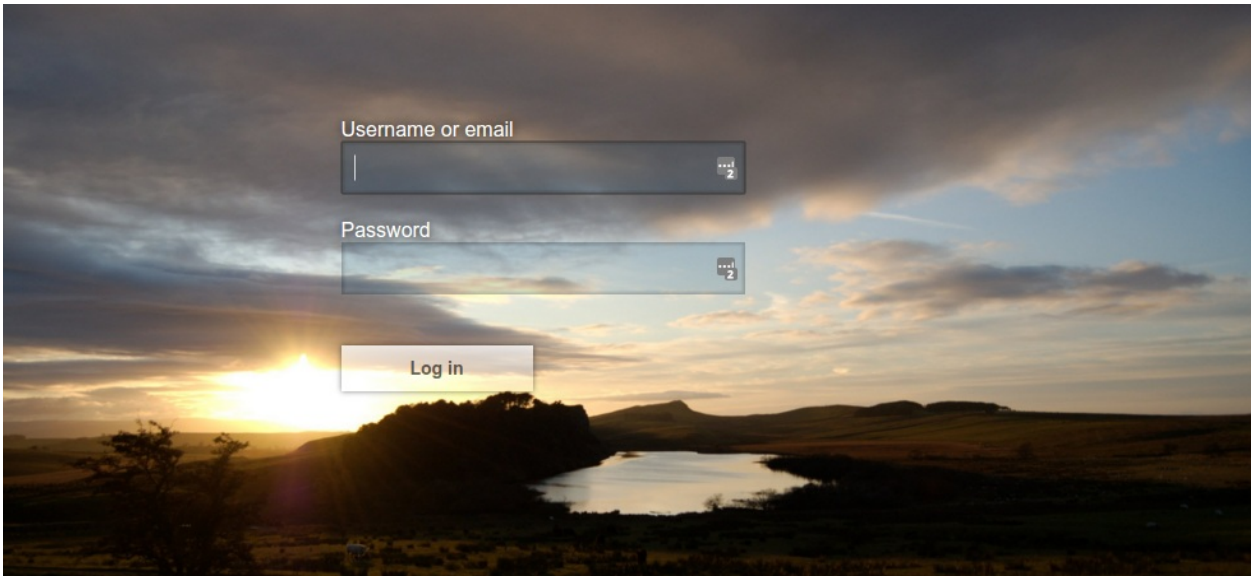
The following example shows how to get the details of the master realm:

```
curl \
  -H "Authorization: bearer eyJhbGciOiJSUz..." \
  "http://localhost:8080/auth/admin/realms/master"
```

CHAPTER 3. THEMES

Red Hat Single Sign-On provides theme support for web pages and emails. This allows customizing the look and feel of end-user facing pages so they can be integrated with your applications.

Figure 3.1. Login page with sunrise example theme



3.1. THEME TYPES

A theme can provide one or more types to customize different aspects of Red Hat Single Sign-On. The types available are:

- Account - Account management
- Admin - Admin console
- Email - Emails
- Login - Login forms
- Welcome - Welcome page

3.2. CONFIGURE THEME

All theme types, except welcome, is configured through the **`Admin Console`**. To change the theme used for a realm open the **`Admin Console`**, select your realm from the drop-down box in the top left corner. Under **`Realm Settings`** click on **`Themes`**.

Note

To set the theme for the **`master`** admin console you need to set the admin console theme for the **`master`** realm. To see the changes to the admin console refresh the page.

To change the welcome theme you need to edit **keycloak-server.json** (in **standalone/configuration** or **domain/servers/{server name}/configuration**) and add **welcomeTheme** to the theme element, for example:

```
"theme": {
    ...
    "welcomeTheme": "custom-theme",
    ...
}
```

If the server is running you need to restart the server for the changes to the welcome theme to take effect.

3.3. DEFAULT THEMES

Red Hat Single Sign-On comes bundled with default themes in the server's root **themes** directory. To simplify upgrading you should not edit the bundled themes directly. Instead create your own theme that extends one of the bundle themes.

3.4. CREATING A THEME

A theme consists of:

- HTML templates (Freemarker Templates)
- Images
- Message bundles
- Stylesheets
- Scripts
- Theme properties

Unless you plan to replace every single page you should extend another theme. Most likely you will want to extend the Red Hat Single Sign-On theme, but you could also consider extending the base theme if you are significantly changing the look and feel of the pages. The base theme primarily consists of HTML templates and message bundles, while the Red Hat Single Sign-On theme primarily contains images and stylesheets.

When extending a theme you can override individual resources (templates, stylesheets, etc.). If you decide to override HTML templates bear in mind that you may need to update your custom template when upgrading to a new release.

While creating a theme it's a good idea to disable caching as this makes it possible to edit theme resources directly from the **themes** directory without restarting Red Hat Single Sign-On. To do this edit **standalone/configuration/keycloak-server.json** for **theme** set **staticMaxAge** to **-1** and both **cacheTemplates** and **cacheThemes** to **false**:

```
"theme": {
    ...
    "staticMaxAge": -1,
    "cacheTemplates": false,
```

```
    "cacheThemes": false,
    ...
}
```

Remember to re-enable caching in production as it will significantly impact performance.

To create a new theme start by creating a new directory in the **themes** directory. The name of the directory becomes the name of the theme. For example to create a theme called **mytheme** create the directory **themes/mytheme**.

Inside the theme directory create a directory for each of the types your theme is going to provide. For example to add the login type to the **mytheme** theme create the directory **themes/mytheme/login**.

For each type create a file **theme.properties** which allows setting some configuration for the theme. For example to configure the theme **themes/mytheme/login** that we just created to extend the base theme and import some common resources create the file **themes/mytheme/login/theme.properties** with following contents:

```
parent=base
import=common/keycloak
```

You have now created a theme with support for the login type. To check that it works open the admin console. Select your realm and click on **Themes**. For **Login Theme** select **mytheme** and click **Save**. Then open the login page for the realm.

You can do this either by login through your application or by opening the Account Management console (**/realms/{realm name}/account**).

To see the effect of changing the parent theme, set **parent=keycloak** in **theme.properties** and refresh the login page.

3.4.1. Theme Properties

Theme properties are set in the file **<THEME TYPE>/theme.properties** in the theme directory.

- parent - Parent theme to extend
- import - Import resources from another theme
- styles - Space-separated list of styles to include
- locales - Comma-separated list of supported locales

There are a list of properties that can be used to change the css class used for certain element types. For a list of these properties look at the theme.properties file in the corresponding type of the keycloak theme (**themes/keycloak/<THEME TYPE>/theme.properties**).

You can also add your own custom properties and use them from custom templates.

3.4.2. Stylesheets

A theme can have one or more stylesheets, to add a stylesheet create a file in the **<THEME TYPE>/resources/css** directory of your theme. Then add it to the **styles** property in **theme.properties**.

For example to add **styles.css** to the **mytheme** create **themes/mytheme/login/resources/css/styles.css** with the following content:

```
.login-pf body {
    background: DimGrey none;
}
```

Then edit **themes/mytheme/login/theme.properties** and add:

```
styles=css/styles.css
```

To see the changes open the login page for your realm. You will notice that the only styles being applied are those from your custom stylesheet. To include the styles from the parent theme you need to load the styles from that theme as well. Do this by editing **themes/mytheme/login/theme.properties** and changing **styles** to:

```
styles=lib/patternfly/css/patternfly.css lib/zocial/zocial.css
css/login.css css/styles.css
```

Note

To override styles from the parent stylesheets it's important that your stylesheet is listed last.

3.4.3. Scripts

A theme can have one or more scripts, to add a script create a file in the **<THEME TYPE>/resources/js** directory of your theme. Then add it to the **scripts** property in **theme.properties**.

For example to add **script.js** to the **mytheme** create **themes/mytheme/login/resources/js/script.js** with the following content:

```
alert('Hello');
```

Then edit **themes/mytheme/login/theme.properties** and add:

```
scripts=js/script.js
```

3.4.4. Images

To make images available to the theme add them to the **<THEME TYPE>/resources/img** directory of your theme. These can be used from within stylesheets or directly in HTML templates.

For example to add an image to the **mytheme** copy an image to **themes/mytheme/login/resources/img/image.jpg**.

You can then use this image from within a custom stylesheet with:

```
body {
    background-image: url('../img/image.jpg');
    background-size: cover;
}
```

Or to use directly in HTML templates add the following to a custom HTML template:

```
<img src="${url.resourcesPath}/img/image.jpg">
```

3.4.5. Messages

Text in the templates are loaded from message bundles. A theme that extends another theme will inherit all messages from the parents message bundle and you can override individual messages by adding **`<THEME TYPE>/messages/messages_en.properties`** to your theme.

For example to replace **`Username`** on the login form with **`Your Username`** for the **`mytheme`** create the file **`themes/mytheme/login/messages/messages_en.properties`** with the following content:

```
usernameOrEmail=Your Username
```

Within a message values like **`{0}`** and **`{1}`** are replaced with arguments when the message is used. For example **`{0} in `Log in to {0}`** is replaced with the name of the realm.

3.4.6. Internationalization

Red Hat Single Sign-On supports internationalization. To enable internationalization for a realm see Server Administration Guide. This section will describe how you can add your own language.

To add a new language create the file **`<THEME TYPE>/messages/messages_<LOCALE>`** in the directory of your theme. Then add it to the **`locales`** property in **`<THEME TYPE>/theme.properties`**. For a language to be available to users the realms**`login`**, **`account`** and **`email`** theme has to support the language, so you need to add your language for those theme types.

For example to add Norwegian translations to the **`mytheme`** theme create the file **`themes/mytheme/login/messages/messages_no.properties`** with the following content:

```
usernameOrEmail=Brukernavn
password=Passord
```

All messages you don't provide a translation for will use the default English translation.

Then edit **`themes/mytheme/login/theme.properties`** and add:

```
locales=en,no
```

You also need to do the same for the **`account`** and **`email`** theme types. To do this create **`themes/mytheme/account/messages/messages_no.properties`** and **`themes/mytheme/email/messages/messages_no.properties`**. Leaving these files empty will result in the English messages being used. Then copy

themes/mytheme/login/theme.properties to **themes/mytheme/account/theme.properties** and **themes/mytheme/email/theme.properties**.

Finally you need to add a translation for the language selector. This is done by adding a message to the English translation. To do this add the following to **themes/mytheme/account/messages/messages_en.properties** and **themes/mytheme/login/messages/messages_en.properties**:

```
locale_no=Norsk
```

3.4.7. HTML Templates

Red Hat Single Sign-On uses Freemarker Templates in order to generate HTML. You can override individual templates in your own theme by creating **<THEME TYPE>/<TEMPLATE>.ftl**. For a list of templates used see **themes/base/<THEME TYPE>**.

When creating a custom template it is a good idea to copy the template from the base theme to your own theme, then applying the modifications you need. Bear in mind when upgrading to a new version of Red Hat Single Sign-On you may need to update your custom templates to apply changes to the original template if applicable.

For example to create a custom login form for the **mytheme** theme copy **themes/base/login/login.ftl** to **themes/mytheme/login** and open it in an editor. After the first line (<#import …>) add **<h1>HELLO WORLD!</h1>** like so:

```
<#import "template.ftl" as layout>
<h1>HELLO WORLD!</h1>
...
```

Check out the FreeMarker Manual for more details on how to edit templates.

3.4.8. Emails

To edit the subject and contents for emails, for example password recovery email, add a message bundle to the **email** type of your theme. There's three messages for each email. One for the subject, one for the plain text body and one for the html body.

To see all emails available take a look at **themes/base/email/messages/messages_en.properties**.

For example to change the password recovery email for the **mytheme** theme create **themes/mytheme/email/messages/messages_en.properties** with the following content:

```
passwordResetSubject=My password recovery
passwordResetBody=Reset password link: {0}
passwordResetBodyHtml=<a href="{0}">Reset password</a>
```

3.5. DEPLOYING THEMES

Themes can be deployed to Red Hat Single Sign-On by copying the theme directory to **themes** or it can be deployed as an archive. During development copying the theme to the **themes** directory, but in production you may want to consider using an **archive**. An **archive** makes it simpler to have a

versioned copy of the theme, especially when you have multiple instances of Red Hat Single Sign-On for example with clustering.

To deploy a theme as an archive you need to create a ZIP archive with the theme resources. You also need to add a file **`META-INF/keycloak-themes.json`** to the archive that lists the available themes in the archive as well as what types each theme provides.

For example for the **`mytheme`** theme create **`mytheme.zip`** with the contents:

- META-INF/keycloak-themes.json
- theme/mytheme/login/theme.properties
- theme/mytheme/login/login.ftl
- theme/mytheme/login/resources/css/styles.css
- theme/mytheme/login/resources/img/image.png
- theme/mytheme/login/messages/messages_en.properties
- theme/mytheme/email/messages/messages_en.properties

The contents of **`META-INF/keycloak-themes.json`** in this case would be:

```
{
    "themes": [{
        "name" : "mytheme",
        "types": [ "login", "email" ]
    }]
}
```

A single archive can contain multiple themes and each theme can support one or more types.

The deploy the archive to Red Hat Single Sign-On you can either manually create a module in **`modules`** or use the **`jboss-cli`** command. It's simplest to use **`jboss-cli`** as it creates the required directories and module descriptor for you.

To deploy **`mytheme.zip`** on Linux run:

```
bin/jboss-cli.sh --command="module add --name=org.example.mytheme --
resources=mytheme.zip"
```

On Windows run:

```
bin\jboss-cli.bat --command="module add --name=org.example.mytheme --
resources=mytheme.zip"
```

This command creates **`modules/org/example/mytheme/main`** directory with the **`mytheme.zip`** archive and **`module.xml`**.

To manually create the module create the directory **`modules/org/keycloak/example/mytheme/main`**, copy **`mytheme.zip`** to this directory and create the file **`modules/org/keycloak/example/mytheme/main/module.xml`** with the contents:

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.3" name="org.keycloak.example.themes">
```

```
    <resources>
        <resource-root path="mytheme.zip"/>
    </resources>
</module>
```

You also need to register the module with Red Hat Single Sign-On. This is done by editing **`keycloak-server.json`** (in **`standalone/configuration`** or **`domain/servers/{server name}/configuration`**) and adding the module to **`theme/module/modules`**. For example:

```
[
"theme": {
    ...
    "module": {
        "modules": [ "org.example.mytheme" ]
    }
}
```

If the server is running you need to restart the server after changing **`keycloak-server.json`**.

Note

If the same theme is deployed to both the **`themes`** directory and as a module the version in the **`themes`** directory is used.

CHAPTER 4. CUSTOM USER ATTRIBUTES

You can add custom user attributes to the registration page and account management console with a custom theme. This chapter describes how to add attributes to a custom theme, but you should refer to the Themes chapter on how to create a custom theme.

4.1. REGISTRATION PAGE

To be able to enter custom attributes in the registration page copy the template **`themes/base/login/register.ftl`** to the login type of your custom theme. Then open the copy in an editor.

As an example to add a mobile number to the registration page add the following snippet to the form:

```
<div class="form-group">
   <div class="${properties.kcLabelWrapperClass!}">
       <label for="user.attributes.mobile"
class="${properties.kcLabelClass!}">Mobile number</label>
   </div>

   <div class="col-sm-10 col-md-10">
       <input type="text" class="${properties.kcInputClass!}"
id="user.attributes.mobile" name="user.attributes.mobile"/>
   </div>
</div>
```

To see the changes make sure your realm is using your custom theme for the login theme and open the registration page.

4.2. ACCOUNT MANAGEMENT CONSOLE

To be able to manage custom attributes in the user profile page in the account management console copy the template **`themes/base/account/account.flt`** to the account type of your custom theme. Then open the copy in an editor.

As an example to add a mobile number to the account page add the following snippet to the form:

```
<div class="form-group">
   <div class="col-sm-2 col-md-2">
       <label for="user.attributes.mobile" class="control-label">Mobile
number</label>
   </div>

   <div class="col-sm-10 col-md-10">
       <input type="text" class="form-control"
id="user.attributes.mobile" name="user.attributes.mobile"
value="${(account.attributes.mobile!'')?html}"/>
   </div>
</div>
```

To see the changes make sure your realm is using your custom theme for the account theme and open the user profile page in the account management console.