Self-Service Reference

/ ForgeRock Identity Management 7.1

Latest update: 7.1.6

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Reference documentation for ForgeRock® Identity Management Self-Service.

Table of Contents

Overview

Reference documentation for the ForgeRock® Identity Management Self-Service REST API.

Note

This guide is reference documentation for IDM's self-contained service. If you are using the platform-based service using trees, see the Platform Self-Service Guide instead.

If you are just getting started, we recommend the platform-based version of self-service.

Quick Start

Self-Service Overview Understand Self-Service Processes	Self-Registration Configure User Self-Registration	Social Registration Configure Registration Using Social Identity Providers
Progressive Profile Progressive Profile Completion	Password Reset Password Reset Process	Username Retrieval Configure Username Retrieval
Additional Configuration Additional configuration options for additional features such as reCAPTCHA, notifications, and the End User UI	Custom Stages Add a Custom Stage to Self-Service	Stage Reference Reference appendix of available self-service stages

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 is intended for anyone developing a self-service application that acts as a client of ForgeRock Identity Management (IDM).

This guide is written with the expectation that you already have basic familiarity with the following topics:

- REST APIs
- JavaScript Object Notation (JSON) and basic IDM configuration

Chapter 1
About User Self-Service

IDM provides a sample End User UI that implements a number of self-service processes, such as self-registration and password reset, based on a Self-Service REST API.

Self-service processes are configured in files named `selfservice-process-name.json` in your project's `conf` directory. Every self-service process steps through a series of *stages*, each with its own requirements, until the end of the process is reached or until the process exits with an exception. The flow through the stages differs, depending on how you have configured the process.

You can customize the default processes, or write your own custom processes by implementing the stages described in "*Self-Service Stage Reference*". For information about how self-service is implemented in the default End User UI, see "Self-Service End User UI". For information on how to customize the End User UI, see the following Git repository: *Identity Management (End User) - UI*.

The Self-Service REST API supports only two HTTP requests:

- `GET` which obtains the requirements for that stage
- `POST` with `_action=submitRequirements`

The response to the `POST` request instructs the client how to proceed. The response can have one of two outcomes:

- Success—all requirements have been submitted and the process advances to the next stage.
- Failure—the behavior here differs by stage. Certain stages will exit with an exception, others will convert the exception into an error that the client must handle, others will simply return the requirements again.

The Self-Service Process Flow

Each self-service process advances through the stages in the order in which they are listed in the `stageConfigs` array in the process configuration file. The password reset process, for example, might include the following stages:

```
{
    "stageConfigs" : [
        {
            "name": "parameters",
            ...
        },
        {
            "name" : "userQuery",
            ...
        },
        {
            "name" : "validateActiveAccount",
            ...
        },
        {
            "name" : "emailValidation",
            ...
        },
        {
            "name" : "kbaSecurityAnswerVerificationStage",
            ...
        },
        {
            "name" : "resetStage",
            ..
        }
    ],
    ...
}
```

A process definition also includes an optional `snapshotToken` and `storage` parameter, for example:

```
{
    "stageConfigs" : [
        ...
    ],
    "snapshotToken" : {
        "type" : "jwt",
        "jweAlgorithm" : "RSAES_PKCS1_V1_5",
        "encryptionMethod" : "A128CBC_HS256",
        "jwsAlgorithm" : "HS256",
        "tokenExpiry" : 300
    },
    "storage" : "stateless"
}
```

The `snapshotToken` specifies the format of the token that is passed between the client and the server with each request. By default, this is a JWT token, stored statelessly, which means that the state is stored in the client, rather than on the server side. Because some legacy clients cannot handle the long URLs provided in a JWT token, you can store the snapshot token locally, as a `uuid` with the following configuration:

```
{
    ...
    "snapshotToken" : {
        "type" : "uuid"
    },
    "storage" : "local"
}
```

In this case, the 16-character token is stored in the IDM repository, in the `jsonstorage` table. To use this feature, copy `/path/to/openidm/samples/example-configurations/self-service/jsonstore.json` to your project's `conf/` directory. This file stores the configuration for the `uuid` token and includes the following settings:

- `entryExpireSeconds`—the amount of time before the password reset URL expires.
- `cleanupDwellSecondsliteral`—how often the server checks for and expires tokens.

  The value of `cleanupDwellSecondsliteral` should be a fraction of `entryExpireSeconds` so that expiration occurs close to the expected expiration time. The check is performed on a periodic basis.

For more information on the self-service tokens, see "Tokens and User Self-Service".

If you do not include the `snapshotToken` and `storage` in the configuration, the default stateless configuration applies.

When a stage advances, it can optionally insert parameters into the process context or *state* for consumption by stages that occur later in the process. The snapshot token is essentially the state of the stage. It is the container in which `state`, `successAdditions` and other data are stored, and then returned to the client at the end of the process, as an encrypted blob named `token`.

Sample configurations for each default self-service process are available in the `/path/to/openidm/samples/example-configurations/self-service` directory.

Each self service process has a specific endpoint under `openidm/selfservice` with the name of the process; for example `openidm/selfservice/reset` for the Password Reset process. If you create a custom self-service process with a configuration file such as `selfservice-myprocess.json`, you produce an endpoint such as `http://localhost:8080/openidm/selfservice/myprocess`.

All REST actions occur against that endpoint. For example, the following initial GET request against the password reset endpoint returns the requirements for the following stage:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "Accept-API-Version: resource=1.0" \
 --request GET \
 "http://localhost:8080/openidm/selfservice/reset"
{
  "_id": "1",
  "_rev": "-852427048",
  "type": "captcha",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Captcha stage",
    "type": "object",
    "required": [
      "response"
    ],
    "properties": {
      "response": {
        "recaptchaSiteKey": "6LcvE1IUAAAAAA5AI1SZzZJl-AlGvHM_dzUg-0_S",
        "description": "Captcha response",
        "type": "string"
      }
    }
  }
}
```

The default End User UI implements the following processes:

- Self-registration (under the endpoint `selfservice/registration`)
- Social registration (under the endpoint `selfservice/socialUserClaim`)
- Password reset (under the endpoint `selfservice/reset`)
- Forgotten username retrieval (under the endpoint `selfservice/username`)
- Progressive profile completion (under `selfservice/profile`)
- Security question updates (under `selfservice/kbaUpdate`)
- Terms and conditions (under `selfservice/termsAndConditions`)

The remainder of this guide describes each stage, its requirements, and expected responses.

Chapter 2
Self-Registration

This chapter describes the configuration, and the requests and responses for user self-registration.

Quick Start

Configuration Configure User Self-Registration.	Registration Form Configure the User Self-Registration Form	Email Registration Configure Emails for Self-Service Registration
User Preferences Configure Synchronization Filters With User Preferences	Multiple Registration Flows Configure Multiple User Self-Registration Flows	Examples Example Self-Registration REST Requests

Configure User Self-Registration

To set up basic user self-registration, you'll need at least the following configuration files:

`ui-configuration.json`

You can find this file in the default IDM project configuration directory, `openidm/conf`.

To enable self-service registration in the UI, enable the following boolean property in `ui-configuration.json`:

```
"selfRegistration" : true,
```

`selfservice-registration.json`

You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`. This includes the following properties:

- `allInOneRegistration`: determines whether IDM collects all user registration information in one or multiple pages. By default, it's set to true:

```
"allInOneRegistration" : true,
```

- `stageConfigs`: configuration details for the stages included in the self-registration process. While the specific stages included may vary, most processes will include at least:
  - `idmUserDetails`: includes the IDM property for email addresses (`mail`), whether or not registration with social identity providers is enabled, and what data is required from new users, as described in "Configure the User Self-Registration Form".
  - `registrationPreferences`: lists preferences to include as defined in the `managed.json` file. For more information, see "Configure User Preferences".
- `snapshotToken`: configuration details for the token used to store the user's details during the registration process.
- `storage`: determines how a user's details are stored for consumption by later stages in the registration process. By default, this is set to `stateless`.

Depending on how you configure User Self-Registration, you may need to set up additional configuration files, as discussed in "Configure the User Self-Registration Form".

Common components included in self-registration include:

- **Email validation**

  If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json`, see "Configuring Emails for Self-Service Registration".

- **Security questions (KBA)**

  If you have configured security questions, users who self-register must create these questions during registration and answer them during the password reset process. You can also configure the system to force users who have been created during a reconciliation from an external data store to add security questions. The relevant code block is shown here, which includes security questions as a stage in the user self-registration process. For related configuration options, see "Configure Security Questions".

  ```
  {
      "name" : "kbaSecurityAnswerDefinitionStage",
      "kbaConfig" : null
  },
  ```

- **Google ReCAPTCHA**

  If you've activated Google reCAPTCHA for user self-service registration, you'll see the following code block:

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "<siteKey>",
    "recaptchaSecretKey" : "<secretKey>",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

As suggested by the code, you'd substitute the actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, see "Configure Google reCAPTCHA".

- **Terms & Conditions**

  If you've set up Terms & Conditions, users who self-register will have to accept them, based on criteria you create, as discussed in "Terms & Conditions". If you've included Terms & Conditions with user self-registration, you'll see the following code block:

  ```
  {
      "name" : "termsAndConditions"
  },
  ```

  New users will have to manually accept these conditions before they complete the self-registration process.

- **Privacy & Consent**

  If you've configured Privacy & Consent, you'll see a code block with the `consent` name. The following code block includes template Privacy & Consent terms in English (`en`) and French (`fr`):

  ```
  {
      "name" : "consent",
      "consentTranslations" : {
          "en" : "Please consent to sharing your data with whomever we like.",
          "fr" : "Veuillez accepter le partage de vos données avec les services de notre choix."
      }
  },
  ```

  Note

  Substitute Privacy & Consent content that meets the requirements of your legal authorities.

For audit activity data related to user self-registration, see Query the Activity Audit Log in the *Audit Guide*.

Configure Self-Registration From the Admin UI

To configure user self-registration from the Admin UI, select Configure > User Registration, and select Enable User Registration on the page that appears. When you enable self-registration from the Admin UI, IDM will create `selfservice-registration.json` for you, if it is not already present. When enabled, you'll see a pop-up window that specifies User Registration Settings, including the following:

- Identity Resource, typically `managed/user`.
- Identity Email Field, typically `mail` or `email`.
- Success URL for the End User UI; users who successfully log in are redirected to that URL. By default, the success URL is `http://localhost:8080/#dashboard/`.
- Preferences, which set up default marketing preferences for new users. New users can change these preferences during registration, or from the End User UI.
- Advanced Options, Snapshot Token, typically a JSON Web Token (JWT).
- Advanced Options, Token Lifetime, with a default of 300 seconds.

Once active, you'll see three tabs under User Registration in the Admin UI:

- *Registration Form*, as described in "Configure the User Self-Registration Form"
- *Social*, as described in "*Social Registration*"
- *Options*, as described in "*Additional Configuration*"

Managing User Self-Registration Over REST

To display the current user self-registration configuration over REST, run the following command:

```
curl \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --request GET \
 "http://localhost:8080/openidm/config/selfservice/registration"
```

Unless you have disabled file writes, the output will match the contents of your project's `selfservice-registration.json` file. For information on disabling file writes, see "Disabling Automatic Configuration Updates" in the *Security Guide*.

If needed, you can update this configuration by including the desired contents of the file:

```
curl \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --header "Content-Type: application/json" \
 --request PUT \
 --data '{ <Insert file contents here> }' \
 "http://localhost:8080/openidm/config/selfservice/registration"
```

Configure the User Self-Registration Form

During user self-registration, IDM lists the attributes that users see in the user registration form, as defined in the `selfservice-registration.json` file. You can modify the properties shown to users in the `registrationProperties` code block:

```
"registrationProperties" : [
    "userName",
    "givenName",
    "sn",
    "mail"
],
```

If you add a managed user property to the `registrationProperties` code block, IDM includes it in the user self-registration screen.

Alternatively, you can add a managed user property in the Admin UI. Select Configure > User Registration, and add a property under the Registration Form tab. This action also adds a managed user property to the noted code block.

In either case, you can change the order of properties; IDM shows the order you configure in the user self-registration screen.

You can also set up user self-registration via configuration files, as described in the following table:

User Self-Registration Configuration Files

File Name	Description
`external.email.json`	To enable email verification, you must configure an outgoing email server.
`managed.json`	You can customize user self-registration based on entries in this file. To change the labels seen by end users, change the associated `title`.
`policy.json`	For more information, see "Add Custom Policies for Self-Registration and Password Reset".
`selfservice.kba.json`	See "Configure Security Questions".
`selfservice-registration.json`	See "Configure User Self-Registration".
`ui-configuration.json`	See "Configure User Self-Registration".

Configuring Emails for Self-Service Registration

To configure emails for self-service registration, you can add the following code block to the `selfservice-registration.json` file:

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "subject" : "Register new account",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Register new account",
        "fr" : "Créer un nouveau compte"
    },
    "messageTranslations" : {
        "en" : "<h3>This is your registration email.</h3><h4><a href=\"%link%\">Email verification link</
a></h4>",
        "fr" : "<h3>Ceci est votre mail d'inscription.</h3><h4><a href=\"%link%\">Lien de vérification
 email</a></h4>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/registration/"
},
```

As suggested by the code block, it includes default registration email messages in English (`en`) and French (`fr`). The `verificationLink` sent with the email takes users to the IDM self-registration URL.

As noted in "Managing User Self-Registration Over REST", you can make these changes over the following endpoint URI: `/openidm/config/selfservice/registration`

If desired, you can also configure self-service registration emails through the Admin UI. Select Configure > User Registration. If needed, activate the Enable User Registration Option. Under the Options tab, in the Email Validation box, select the ✎ icon. The Configure Validation Email pop-up should appear.

When you use the Admin UI to customize self-registration emails, you can review the changes in the `selfservice-registration.json` file.

Configure User Preferences

You can set up preferences for managed users, such as those related to marketing and news updates. You can then use those preferences as a filter when reconciling users to a target repository.

In the default project, common marketing preference options are included for the managed user object. To find these preferences in the Admin UI, select Configure > Managed Objects and select the User managed object. Under the Preferences tab, you'll see keys and descriptions. You can also see these preferences in the `managed.json` file, illustrated here:

```
"preferences" : {
    "title" : "Preferences",
    "description" : "Preferences",
    "viewable" : true,
    "searchable" : false,
    "userEditable" : true,
    "type" : "object",
    "usageDescription" : "",
    "isPersonal" : false,
    "properties" : {
        "updates" : {
            "description" : "Send me news and updates",
            "type" : "boolean"
        },
        "marketing": {
            "description" : "Send me special offers and services",
            "type" : "boolean"
        }
    },
    "order": [
        "updates",
        "marketing"
    ],
    "required": []
},
```

Reviewing Preferences as an End User

When regular users log in to the End User UI, they'll see the preferences described in "Configure User Preferences". When they accept the preferences, their managed user objects are updated with entries similar to the following:

```
"preferences" : {
    "updates" : true,
    "marketing" : true
},
```

User Preferences and Reconciliation

You can configure user preferences as a filter for reconciliation. For example, if some of your users do not want marketing emails, you can filter those users out of any reconciliation operation.

1. To configure user preferences as a filter, log in to the Admin UI.

2. Select Configure > Mappings. Choose a mapping.

3. Under the Association tab, select Individual Record Validation.

4. Based on the options in the Valid Source drop-down list, you can select `Validate based on user preferences`. Users who have selected a preference such as `Send me special offers` will then be reconciled from the source to the target repository.

Note

What IDM does during this reconciliation depends on the policy associated with the `UNQUALIFIED` situation for a `validSource`. The default action is to delete the target object (user). For more information, see "How Synchronization Situations Are Assessed" in the *Synchronization Guide*.

Alternatively, edit the mapping file directly. The following excerpt of a mapping file includes `preferences` as conditions to define a `validSource` on an individual record validation. IDM applies these conditions at the next reconciliation.

```
"validSource" : {
    "type" : "text/javascript",
    "globals" : {
        "preferences" : [
            "updates",
            "marketing"
        ]
    },
    "file" : "ui/preferenceCheck.js"
},
"validTarget" : {
    "type" : "text/javascript",
    "globals" : { },
    "source" : ""
}
```

Configure Multiple User Self-Registration Flows

You can set up multiple self-registration flows, with features limited only by the capabilities listed in "*Self-Registration*".

Note

Multiple self-registration flows, and customization of the End User UI beyond what is described in this document (and the noted public Git repository), are *not* supported.

For additional information on customizing the End User UI, see the following ForgeRock Git repository: *ForgeRock/end-user-ui: Identity Management (End User)*.

For example, you may want to set up different portals for regular employees and contractors. You'd configure each portal with different self-registration flows, managed by the same IDM backend. Each portal would use the appropriate registration API.

To prepare for this section, you'll need a `selfservice-registration.json` file. You can find a copy in the following directory: `/path/to/openidm/samples/example-configurations/self-service`.

To avoid errors when using this file, you should either:

- Copy the following files from the same directory:

  `selfservice.terms.json`
  `selfservice-termsAndConditions.json`

- Delete the `termsAndConditions` code block from the respective `selfservice-registration*.json` files.

User self-registration is normally coded in the `selfservice-registration.json` file. In preparation, copy this file to the `selfservice-registration*.json` to the names shown in the following list:

- Employee Portal
  - Configuration file: `selfservice-registrationEmployee.json`
  - URL: `https://localhost:8443/openidm/selfservice/registrationEmployee`
  - `verificationLink`: `https://localhost:8443/#/registrationEmployee`
- Contractor Portal
  - Configuration file: `selfservice-registrationContractor.json`
  - URL: `https://localhost:8443/openidm/selfservice/registrationContractor`
  - `verificationLink`: `https://localhost:8443/#/registrationContractor`

Edit the configuration file for each portal.

1. Modify the `verificationLink` URL associated with each portal as described.
2. Edit your access configuration (`conf/access.json`), by adding an endpoint for each new self-service registration file, after the `selfservice/registration` section. For example, the following code excerpt would apply to the `registrationEmployee` and `registrationContractor` endpoints:

```
{
    "pattern"    : "selfservice/registrationEmployee",
    "roles"      : "*",
    "methods"    : "read,action",
    "actions"    : "submitRequirements"
},
{
    "pattern"    : "selfservice/registrationContractor",
    "roles"      : "*",
    "methods"    : "read,action",
    "actions"    : "submitRequirements"
},
```

3. Modify the functionality of each selfservice-registration*.json file as desired. For guidance, see the sections noted in the following table:

Configuring `selfservice-registration*.json` Files for Different Portals

Feature	Code Block	Link
Social Registration	`"socialRegistrationEnabled" : true,`	"*Social Registration*"
Properties requested during self-registration	`"registrationProperties" : [ "userName", "givenName", "sn", "mail" ],`	"Configure the User Self-Registration Form"
Terms & Conditions	`{ "name" : "termsAndConditions" }`	"Terms & Conditions"
Privacy & Consent	`{ "name" : "consent", "consentTranslations" : { "en" : "substitute appropriate Privacy & Consent wording", "fr" : "substitute appropriate Privacy & Consent wording, in French" } },`	
reCAPTCHA	`{ "name" : "captcha", "recaptchaSiteKey" : "<siteKey>", "recaptchaSecretKey" : "<secretKey>", "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify" }`	"Configure Google reCAPTCHA"
Email Validation		"Configuring Emails for Self-Service Registration"
Security Questions	`{ "name" : "kbaSecurityAnswerDefinitionStage", "kbaConfig" : null },`	"Configure Security Questions"

If you leave out the code blocks associated with the feature, you won't see that feature in the self-service registration flow. In that way, you can set up different self-service registration flows for the Employee and Contractor portals.

Once you've configured both portals, you can make REST calls to both URLs:

```
https://localhost:8443/openidm/selfservice/registrationEmployee
https://localhost:8443/openidm/selfservice/registrationContractor
```

For more advice on how you can create custom registration flows, see the following public ForgeRock Git repository: *Identity Management (End User) - UI*.

Note

The changes described in this section require changes to the End User UI source code as described in the noted public Git repository. Pay particular attention to the instructions associated with the `Registration.vue` file.

Example Self-Registration REST Requests

The REST calls shown in this chapter assume that user registration is enabled with the default security questions, and that the configuration is similar to that shown in the sample registration configuration file (`samples/example-configurations/self-service/selfservice-registration.json`):

+ *Example Self-Registration Configuration*

```
{
    "allInOneRegistration" : true,
    "stageConfigs" : [
        {
            "name": "parameters",
            "parameterNames" : [
                "returnParams"
            ]
        },
        {
            "name" : "idmUserDetails",
            "identityEmailField" : "mail",
            "socialRegistrationEnabled" : true,
            "identityServiceUrl" : "managed/user",
            "registrationProperties" : [
                "userName",
                "givenName",
                "sn",
                "mail"
            ],
            "registrationPreferences": ["marketing", "updates"]
        },
        {
            "name" : "termsAndConditions"
        },
        {
            "name" : "emailValidation",
            "identityEmailField" : "mail",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@admin.org",
            "subject" : "Register new account",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Register new account",
                "fr" : "Créer un nouveau compte"
            },
            "messageTranslations" : {
                "en" : "<h3>This is your registration email.</h3><h4><a href=\"%link%\">Email
 verification link</a></h4>",
                "fr" : "<h3>Ceci est votre email d'inscription.</h3><<h4><a href=\"%link%\">Lien de
 vérification email</a></h4>"
            },
            "verificationLinkToken" : "%link%",
            "verificationLink" : "https://idm.example.com:8443/#/registration/"
        },
        {
            "name" : "kbaSecurityAnswerDefinitionStage",
            "kbaConfig" : null
        },
        {
            "name" : "selfRegistration",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "localAutoLogin",
            "successUrl" : "",
            "identityUsernameField": "userName",
            "identityPasswordField": "password"
        }
    ],
    "storage" : "stateless"
}
```

1. The client loads the initial registration form. The server returns the `initial` tag to indicate the start of the registration process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request GET \
 "https://idm.example.com:8443/openidm/selfservice/registration"
{
  "_id": "1",
  "_rev": "1113597344",
  "type": "parameters",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Parameters",
    "type": "object",
    "properties": {
      "returnParams": {
        "description": "Parameter named 'returnParams'",
        "type": "string"
      }
    }
  }
}
```

The client sends an empty POST request with the `submitRequirements` action.

The server returns the following:

- The `initial` tag to indicate the start of the registration process.
- A `token` that must be provided in subsequent steps.
- A JSON `requirements` object that must be provided in subsequent steps.

+ *Example Registration Submission*

```
curl \
 --header "Content-type: application/json" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request POST \
 --data '{"input":{"input":{}}}' \
 https://idm.example.com:8443/openidm/selfservice/registration?_action=submitRequirements
{
  "type":"allInOneRegistration",
  "tag":"initial",
  "requirements":{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "description":"All-In-One Registration",
    "type":"object",
    "properties":{
      "response":{
        "recaptchaSiteKey":"6Lf...1ry",
        "description":"Captcha response",
        "type":"string"
      },
      "kba":{
        "type":"array",
        "minItems":2,
        "items":{
          "type":"object",
          "oneOf":[
            {
              "$ref":"#/definitions/systemQuestion"
            },
            {
              "$ref":"#/definitions/userQuestion"
            }
          ]
        },
        "questions":[
          {
            "question":{
              "en":"What's your favorite color?",
              "en_GB":"What is your favourite colour?",
              "fr":"Quelle est votre couleur préférée?"
            },
            "id":"1"
          },
          {
            "question":{
              "en":"Who was your first employer?"
            },
            "id":"2"
          }
        ]
      },
      "user":{
        "default":{
        },
        "description":"User Object",
        "type":"object"
      },
      "accept":{
        "description":"Accept",
        "type":"string"
      }
    },
    "required":[
      "response",
      "accept",
      "kba"
    ],
    "terms":"These are our terms and conditions",
    "termsVersion":"1.0",
    "uiConfig":{
      "displayName":"We have updated our terms",
      "purpose":"To proceed, accept these terms",
      "buttonText":"Accept"
    },
    "createDate":"2018-11-05T13:14:00.540Z",
    "definitions":{
      "systemQuestion":{
        "description":"System Question",
        "type":"object",
        "required":[
          "questionId",
          "answer"
        ],
        "properties":{
          "questionId":{
            "description":"Id of predefined question",
            "type":"string"
          },
          "answer":{
            "description":"Answer to the referenced question",
            "type":"string"
          }
        },
        "additionalProperties":false
      },
      "userQuestion":{
        "description":"User Question",
        "type":"object",
        "required":[
          "customQuestion",
          "answer"
          ],
          "properties":{
            "answer":{
               "description":"Answer to the question",
               "type":"string"
            },
            "customQuestion":{
               "description":"Question defined by the user",
               "type":"string"
            }
          },
          "additionalProperties":false
        },
        "providers":{
          "type":"array",
          "items":{
            "type":"object",
            "oneOf":[
            ]
          }
        }
      },
      "socialRegistrationEnabled":false,
      "registrationForm":null,
      "registrationProperties":{
        "properties":{
          "userName":{
            "title":"Username",
            "description":"Username",
            "viewable":true,
            "type":"string",
            "searchable":true,
            "userEditable":true,
            "usageDescription":"",
            "isPersonal":true,
            "policies":[
              {
                "policyId" : "minimum-length",
                "params" : {
                  "minLength" : 1
                }
              },
              {
                "policyId":"unique"
              },
              {
                "policyId":"no-internal-user-conflict"
              },
              {
                "policyId":"cannot-contain-characters",
                "params":{
                  "forbiddenChars":[
                    "/"
                  ]
                }
              }
            ]
          },
          "givenName":{
            "title":"First Name",
            "description":"First Name",
            "viewable":true,
            "type":"string",
            "searchable":true,
            "userEditable":true,
            "usageDescription":"",
            "isPersonal":true
          },
          "sn":{
            "title":"Last Name",
            "description":"Last Name",
            "viewable":true,
            "type":"string",
            "searchable":true,
            "userEditable":true,
            "usageDescription":"",
            "isPersonal":true
          },
          "mail":{
            "title":"Email Address",
            "description":"Email Address",
            "viewable":true,
            "type":"string",
            "searchable":true,
            "userEditable":true,
            "usageDescription":"",
            "isPersonal":true,
            "policies":[
              {
                 "policyId":"valid-email-address-format"
              }
            ]
          }
        },
        "required":[
          "userName",
          "givenName",
          "sn",
          "mail"
        ]
      },
      "registrationPreferences":{
        "updates":{
          "description":"Send me news and updates",
          "type":"boolean"
        },
        "marketing":{
          "description":"Send me special offers and services",
          "type":"boolean"
        }
      },
      "stages":[
        "captcha",
        "termsAndConditions",
        "kbaSecurityAnswerDefinitionStage",
        "idmUserDetails"
      ]
    },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQ...2h-k"
}
```

2. The client sends a POST request with the requirements. The server responds with a request for the emailed code:

```
curl \
 --header "Content-type: application/json" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request POST \
 --data '{
  "input":{
    "user":{
      "userName":"bjensen",
      "givenName":"Babs",
      "sn":"Jensen",
      "mail":"babs.k.jensen@gmail.com",
      "preferences":{
        "updates":false,
        "marketing":false
      },
      "password":"Passw0rd"
    },
    "kba":[
      {
        "answer":"red",
        "questionId":"1"
      },
      {
        "answer":"forgerock",
        "questionId":"2"
      }
    ],
    "response":"03AMGVjXggloUomtJx2Q0_wAjzyb9lN3LJBRIN67O85eGJIejO6WMlZGZ2jqnz...",
    "g-recaptcha-response":"03AMGVjXggloUomtJx2Q0_wAjzyb9lN3LJBRIN67O85eGJIejO...",
    "accept":"true"
  },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQWlPa..."
}' \
https://idm.example.com:8443/openidm/selfservice/registration?_action=submitRequirements
{
  "type":"emailValidation",
  "tag":"validateCode",
  "requirements":{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "description":"Verify emailed code",
    "type":"object",
    "required":[
      "code"
    ],
    "properties":{
      "code":{
        "description":"Enter code emailed",
        "type":"string"
      }
    }
  },
  "token":"eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVYQWl..."
}
```

Note

By default, the snapshot token expires after 300 seconds. If the delay between the first request and the second request is greater than that period, the snapshot token will be invalid and the initial request must be sent again to obtain a fresh snapshot token. You can change the snapshot token expiration time in the self-service process configuration file (`selfservice-registration.json` in this case).

The following excerpt of the configuration file shows the default `snapshotToken` configuration. To change the expiration time, set the `tokenExpiry` property:

```
"snapshotToken" : {
    "type" : "jwt",
    "jweAlgorithm" : "RSAES_PKCS1_V1_5",
    "encryptionMethod" : "A128CBC_HS256",
    "jwsAlgorithm" : "HS256",
    "tokenExpiry" : 300
},
```

3. The email verification link redirects to:

```
https://idm.example.com:8443/#/registration/&token=eyJ0e..."
```

The client is registered and logged into the End User UI.

Chapter 3
Social Registration

IDM provides a standards-based solution for social authentication requirements, based on the OAuth 2.0 and OpenID Connect 1.0 standards. They are similar, as OpenID Connect 1.0 is an authentication layer built on OAuth 2.0.

This chapter describes how to configure IDM to register and authenticate users with multiple social identity providers.

To configure different social identity providers, you'll take the same general steps:

- Set up the provider. You'll need information such as a `Client ID` and `Client Secret` to set up an interface with IDM.
- Configure the provider on IDM.
- Set up User Registration. Activate `Social Registration` in the applicable Admin UI screen or configuration file.
- After configuration is complete, test the result. For a common basic procedure, see "Test Social Identity Providers".

You can configure how IDM handles authentication using social identity providers by opening the Admin UI and selecting Configure > Authentication > Modules > Social Providers. The Social Providers authentication module is enabled by default. For more information, see "Configure the Social Providers Authentication Module".

To understand how data is transmitted between IDM and a social identity provider, read "OpenID Connect Authorization Code Flow".

Note

For all social identity providers, set up a FQDN for IDM, along with information in a DNS server, or system `hosts` files. For test purposes, FQDNs that comply with RFC 2606, such as `localhost` and `openidm.example.com`, are acceptable.

When you've configured one or more social identity providers, you can activate the Social Registration option in User Registration. This action adds:

- The following setting to the `selfservice-registration.json` configuration file:

  ```
  "socialRegistrationEnabled" : true,
  ```

- The following configuration file: `selfservice-socialUserClaim.json`, discussed in "Account Claiming: Links Between Accounts and Social Identity Providers".

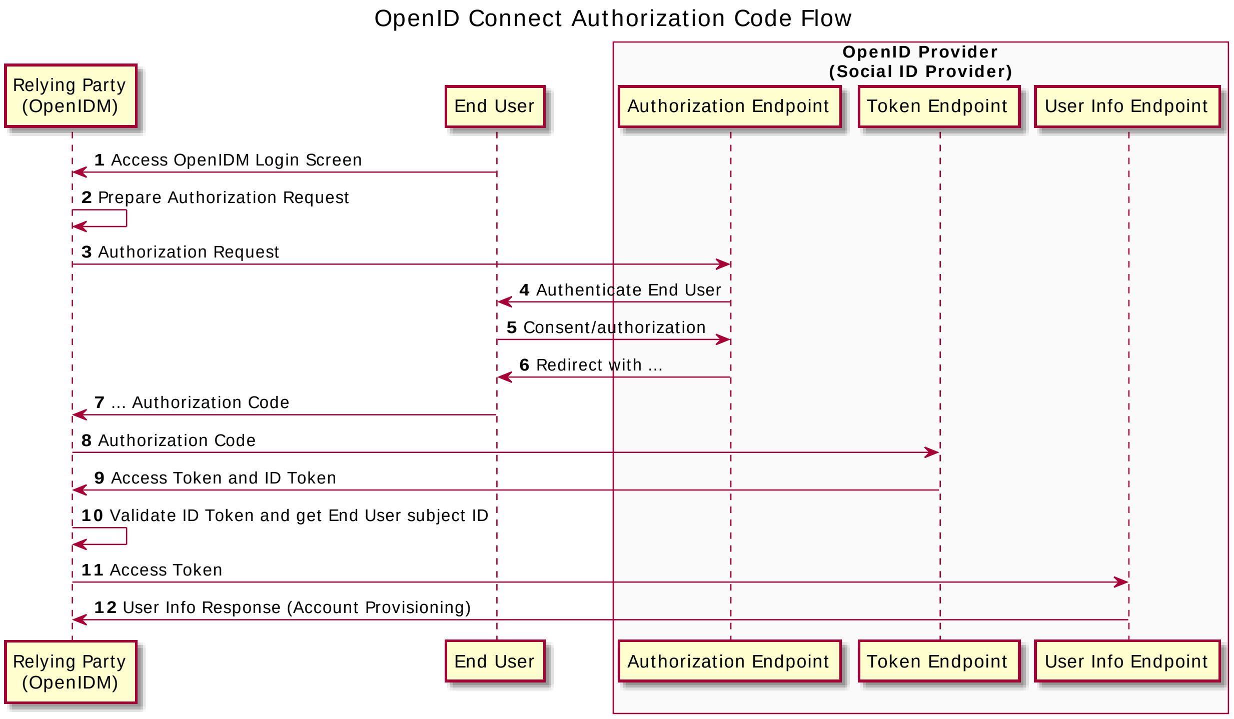
Under the Social tab, you'll see a list of property mappings as defined in the `selfservice.propertymap.json` file.

One or more `source` properties in this file takes information from a social identity provider. When a user registers with their social identity account, that information is reconciled to the matching `target` property for IDM. For example, the `email` property from a social identity provider is normally reconciled to the IDM managed user `mail` property.

OpenID Connect Authorization Code Flow

The OpenID Connect Authorization Code Flow specifies how IDM (Relying Party) interacts with the OpenID Provider (Social ID Provider), based on the use of the OAuth 2.0 authorization grant. The following sequence diagram illustrates successful processing from the authorization request, through grant of the authorization code, access token, ID token, and provisioning from the social identity provider to IDM.

OpenID Connect Authorization Code Flow for Social ID Providers


OpenID Connect Authorization Code Flow
OpenID Provider (Social ID Provider)
Relying Party (OpenIDM)
End User
Authorization Endpoint
Token Endpoint
User Info Endpoint
1 Access OpenIDM Login Screen
2 Prepare Authorization Request
3 Authorization Request
4 Authenticate End User
5 Consent/authorization
6 Redirect with ...
7 ... Authorization Code
8 Authorization Code
9 Access Token and ID Token
10 Validate ID Token and get End User subject ID
11 Access Token
12 User Info Response (Account Provisioning)

The following list describes details of each item in the authorization flow:

1. A user navigates to the IDM End User UI, and selects the `Sign In` link for the desired social identity provider.
2. IDM prepares an authorization request.
3. IDM sends the request to the Authorization Endpoint that you configured for the social identity provider, with a Client ID.
4. The social identity provider requests end user authentication and consent.
5. The end user transmits authentication and consent.
6. The social identity provider sends a redirect message, with an authorization code, to the end user's browser. The redirect message goes to an `oauthReturn` endpoint, configured in `ui.context-oauth.json` in your project's `conf/` directory.

   When you configure a social identity provider, you'll find the endpoint in the applicable configuration file with the following property: `redirectUri`.

7. The browser transmits the redirect message, with the authorization code, to IDM.
8. IDM records the authorization code, and sends it to the social identity provider Token Endpoint.
9. The social identity provider token endpoint returns access and ID tokens.
10. IDM validates the token, and sends it to the social identity provider User Info Endpoint.
11. The social identity provider responds with information on the user's account, that IDM can provision as a new Managed User.

You'll configure these credentials and endpoints, in some form, for each social identity provider.

Many Social Identity Providers, One Schema

Most social identity providers include common properties, such as name, email address, icon configuration, and location.

IDM includes two sets of property maps that translate information from a social identity provider to your managed user objects. These property maps are as follows:

- The `identityProviders.json` file includes a `propertyMap` code block for each supported provider. This file maps properties from the provider to a generic managed user object. You should not customize this file. To use this file, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory.
- The `selfservice.propertymap.json` file translates the generic managed user properties to the managed user schema that you have defined in `managed.json`. If you have customized the managed user schema, this is the file that you must change, to indicate how your custom schema maps to the generic managed user schema.

Examine `conf/identityProviders.json`. The following excerpt shows the Facebook `propertyMap`:

```
"propertyMap" : [
   {
      "source" : "id",
      "target" : "id"
   },
   {
      "source" : "name",
      "target" : "displayName"
   },
   {
      "source" : "first_name",
      "target" : "givenName"
   },
   {
      "source" : "last_name",
      "target" : "familyName"
   },
   {
      "source" : "email",
      "target" : "email"
   },
   {
      "source" : "email",
      "target" : "username"
   },
   {
      "source" : "locale",
      "target" : "locale"
   }
]
```

The source lists the Facebook property, the target lists the corresponding property for a generic managed user.

IDM then processes that information through the `selfservice.propertymap.json` file, where the source corresponds to the generic managed user and the target corresponds to your customized managed user schema (defined in your project's `managed.json` file).

```
{
   "properties" : [
      {
         "source" : "givenName",
         "target" : "givenName"
      },
      {
         "source" : "familyName",
         "target" : "sn"
      },
      {
         "source" : "email",
         "target" : "mail"
      },
      {
         "source" : "postalAddress",
         "target" : "postalAddress",
         "condition" : "/object/postalAddress  pr"
      },
      {
         "source" : "addressLocality",
         "target" : "city",
         "condition" : "/object/addressLocality  pr"
      },
      {
         "source" : "addressRegion",
         "target" : "stateProvince",
         "condition" : "/object/addressRegion  pr"
      },
      {
         "source" : "postalCode",
         "target" : "postalCode",
         "condition" : "/object/postalCode  pr"
      },
      {
         "source" : "country",
         "target" : "country",
         "condition" : "/object/country  pr"
      },
      {
         "source" : "phone",
         "target" : "telephoneNumber",
         "condition" : "/object/phone  pr"
      },
      {
         "source" : "username",
         "target" : "userName"
      }
   ]
}
```

Tip

To take additional information from a social identity provider, make sure the property is mapped through the `identityProviders.json` and `selfservice.propertymap.json` files.

Several of the property mappings include a `pr` presence expression which is a filter that returns all records with the given attribute. For more information, see "Presence Expressions" in the *Object Modeling Guide*.

Amazon Social Identity Provider

- "Set Up Amazon"
- "Configure an Amazon Social Identity Provider"
- "Configure User Registration to Link to Amazon"
- "Amazon Social Identity Provider Configuration Details"

Note

Amazon as a social identity provider requires access over secure HTTP (HTTPS).

Set Up Amazon

To set up Amazon as a social identity provider, first Register for Login With Amazon. You will need an Amazon account.

Then, create a security profile. You will need the following information:

- Security Profile Name (The name of your app)
- Security Profile Description
- Consent Privacy Notice URL
- Consent Logo Image (optional)

When complete and saved, you should see a list of security profiles with `OAuth2` credentials. You should be able to find the `Client ID` and `Client Secret` from this screen.

You still need to configure the web settings for your new Security Profile. From the Amazon Developer Console dashboard, select Apps and Services > Login with Amazon, then select Manage > Web Settings.

In the `Web Settings` for your app, you'll need to set either of the following properties:

- Allowed Origins, which should match the URL for your registration page, such as `https://openidm.example.com:8443`
- Allowed Return URLs, which should match the redirect URIs described in "Configure an Amazon Social Identity Provider". You may see URIs such as `https://openidm.example.com:8443/`.

Configure an Amazon Social Identity Provider

1. To configure an Amazon social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Amazon social identity provider.

   In the `Amazon Provider` pop-up that appears, the values for `Redirect URI` should match the values that you've entered for Allowed Return URLs in "Set Up Amazon".
3. Include the values that Amazon created for `Client ID` and `Client Secret`, as described in "Set Up Amazon".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Amazon Social Identity Provider Configuration Details".

When you enable an Amazon social identity provider in the Admin UI, IDM generates a `conf/identityProvider-amazon.json` file.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, endpoints, as well as the values for `clientId` and `clientSecret`.

```
{
   "provider" : "amazon",
   "authorizationEndpoint" : "https://www.amazon.com/ap/oa",
   "tokenEndpoint" : "https://api.amazon.com/auth/o2/token",
   "userInfoEndpoint" : "https://api.amazon.com/user/profile"
   "enabled" : true,
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "profile"
   ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with an Amazon social identity, you can verify this by selecting Manage > Amazon, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Amazon Social Identity Provider Configuration Details".

Configure User Registration to Link to Amazon

Once you've configured the Amazon social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "*Admin UI*" in the *Setup Guide*.

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Amazon Social Identity Provider Configuration Details

You can set up the Amazon social identity provider through the Admin UI or in a `conf/identityProvider-amazon.json` file. IDM generates the `identityProvider-amazon.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Amazon Provider pop-up window, along with associated information in the `identityProvider-amazon.json` file:

Amazon Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Amazon App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Amazon API
Scope	`scope`	An array of strings that allows access to user data; see Amazon's *Customer Profile* Documentation.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://www.amazon.com/ap/oa`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.amazon.com/auth/o2/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.amazon.com/user/profile`
Not in the Admin UI	`name`	Name of the social identity provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between Amazon and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Apple Social Identity Provider

To configure Apple as a social identity provider (Sign in with Apple), you'll need an Apple developer account.

- "Configure Apple Login"
- "Configure an Apple Identity Provider"
- "Configure User Registration through Apple"
- "Apple Social Identity Provider Configuration Details"

Configure Apple Login

You need a client ID and client secret for your application. In the Apple developer portal, the client ID is called a `Services ID`.

1. Log in to the Apple Developer Portal.
2. Select Certificates, Identifiers and Profiles > Identifiers.
3. On the Identifiers page, select Register a New Identifier, then select Services IDs.
4. Enter a Description and Identifier for this Services ID, and make sure that Sign in With Apple is enabled.

   Important

   The Identifier you specify here will be your OAuth Client ID.

5. Click Configure.
6. On the Web Authentication Configuration screen, enter the Web Domain on which IDM runs, and specify the redirect URL used during the OAuth flow (Return URLs).

   The redirect URL must have the following format:

   ```
   https://idm.example.com/redirect
   ```

   Note

   You must use a real domain (FQDN) here. Apple does not allow `localhost` URLs. If you enter an IP address such as `127.0.0.1`, it will fail later in the OAuth flow.

7. Click Save > Continue > Register.
8. Generate the client secret.

   Instead of using simple strings as OAuth client secrets, Apple uses a public/private key pair, where the client secret is a signed JWT. To register the private key with Apple:

   a. Select Certificates, Identifiers and Profiles > Keys, then click the `+` icon to register a new key.

b. Enter a Key Name and enable Sign In with Apple.

c. Click Configure, then select the primary App ID that you created previously.

d. Apple generates a new private key, in a `.p8` file.

> **Caution**
>
> You can only download this key *once*. Ensure that you save this file, because you will not be able to download it again.

Rename the file to `key.txt`, then locate the Key ID in that file.

e. Use this private key to generate a client secret JWT. Sign the JWT with your private key, using an ES256 algorithm.

Configure an Apple Identity Provider

1. To configure an Apple social identity provider, log in to the Admin UI and select Configure > Social ID Providers.
2. Enable the Apple social identity provider.

   In the Apple Provider window, enter the Redirect URI that you set up in "Configure Apple Login".
3. Enter your Client ID and Client Secret.

Configure User Registration through Apple

When you have configured the Apple social identity provider, you can activate it through User Registration.

1. In the Admin UI, select Configure > User Registration > Enable User Registration.
2. On the Social tab, enable Social Registration.

   For more information, see "Self-Service End User UI".

Apple Social Identity Provider Configuration Details

You can set up the Apple social identity provider through the Admin UI or in a `conf/identityProvider-apple.json` file. IDM generates the `identityProvider-apple.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Apple Provider pop-up window, along with associated information in the `identityProvider-apple.json` file.

Apple Social Identity Provider Configuration Properties

Property (UI)	**Property (JSON file)**	**Description**
Client ID	`clientId`	The client identifier for your Apple App. In the Apple developer portal, the client ID is called a `Services ID`.
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Apple API.
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically, `https://appleid.apple.com/auth/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. Typically, `https://appleid.apple.com/auth/token`.
Well-Known Endpoint	`wellKnownEndpoint`	Access for other URIs. Typically, `https://appleid.apple.com/.well-known/openid-configuration`.
Issuer	`issuer`	The token issuer. Typically, `https://appleid.apple.com`.
Not in the Admin UI	`provider`	Name of the social identity provider.
Not in the Admin UI	`configClass`	Configuration class for the authentication module.
Not in the Admin UI	`basicAuth`	Whether to use basic authentication.
Not in the Admin UI	`propertyMap`	Mapping between Apple and IDM.

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Facebook Social Identity Provider

- "Set Up Facebook"
- "Configure a Facebook Social Identity Provider"
- "Configure User Registration to Link to Facebook"
- "Facebook Social Identity Provider Configuration Details"

Note

As of October 2018, Facebook as a social identity provider requires access over secure HTTP (HTTPS).

Set Up Facebook

To set up Facebook as a social identity provider, navigate to the *Facebook for Developers* page. You'll need a Facebook account. While you could use a personal Facebook account, it is best to use an organizational account to avoid problems if specific individuals leave your organization. When you set up a Facebook social identity provider, you'll need to perform the following tasks:

- In the Facebook for Developers page, select My Apps and Add a New App. For IDM, you'll create a `Website` application.
- You'll need to include the following information when creating a Facebook website application:
  - Display Name
  - Contact Email
  - IDM URL
- When complete, you should see your App. Navigate to Basic Settings.
- Make a copy of the `App ID` and `App Secret` for when you configure the Facebook social identity provider in IDM.
- In the settings for your App, you should see an entry for `App Domains`, such as `example.com`, as well as a Website Site URL, such as `https://idm.example.com/`.

For Facebook's documentation on the subject, see *Facebook Login for the Web with the JavaScript SDK*.

Configure a Facebook Social Identity Provider

1. To configure a Facebook social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Facebook social identity provider.
3. Include the values that Facebook created for `App ID` and `App Secret`, as described in "Set Up Facebook".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Facebook Social Identity Provider Configuration Details".

When you enable a Facebook social identity provider in the Admin UI, IDM generates the `identityProvider-facebook.json` file in your project's `conf/` subdirectory.

It includes parts of the file that you may have configured through the Admin UI. While the labels in the UI specify App ID and App Secret, you'll see them as `clientId` and `clientSecret`, respectively, in the configuration file.

```
{
   "provider" : "facebook",
   "authorizationEndpoint" : "https://www.facebook.com/dialog/oauth",
   "tokenEndpoint" : "https://graph.facebook.com/v2.7/oauth/access_token",
   "userInfoEndpoint" : "https://graph.facebook.com/me?
fields=id,name,picture,email,first_name,last_name,locale"
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
             "cipher" : "AES/CBC/PKCS5Padding",
             "stableId" : "openidm-sym-default",
             "salt" : "<hashValue>",
             "data" : "<encryptedValue>",
             "keySize" : 16,
             "purpose" : "idm.config.encryption",
             "iv" : "<encryptedValue>",
             "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "email",
       "user_birthday"
   ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Facebook social identity, you can verify this by selecting Manage > Facebook, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Facebook Social Identity Provider Configuration Details".

Configure User Registration to Link to Facebook

Once you've configured the Facebook social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and under the Social tab, enable the option associated with Social Registration. For more information about user self-service features, see "Self-Service End User UI".

When you enable social registration, you're allowing users to register on IDM through all active social identity providers.

Facebook Social Identity Provider Configuration Details

You can set up the Facebook social identity provider through the Admin UI or in a `conf/identityProvider-facebook.json` file. IDM generates the `identityProvider-facebook.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Facebook Provider pop-up window, along with associated information in the `identityProvider-facebook.json` file:

Facebook Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
App ID	`clientId`	The client identifier for your Facebook App
App Secret	`clientSecret`	Used with the App ID to access the applicable Facebook API
Scope	`scope`	An array of strings that allows access to user data; see Facebook's *Permissions Reference* Documentation.
Authorization Endpoint	`authorizationEndpoint`	For Facebook's implementation, see their documentation on how they *Manually Build a Login Flow*.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. For Facebook's implementation, see their documentation on how they *Manually Build a Login Flow*.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields through Facebook's API. The default endpoint includes the noted field properties as a list, as defined in Facebook's *Permissions Reference*.
Not in the Admin UI	`name`	Name of the Social ID provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between Facebook and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Google Social Identity Provider

- "Set Up Google".
- "Configure a Google Social Identity Provider".
- "Configure User Registration to Link to Google".
- "Google Social Identity Provider Configuration Details".

Set Up Google

To set up Google as a social identity provider, navigate to the *Google API Manager*. You'll need a Google account. If you have GMail, you already have a Google account. While you could use a personal Google account, it is best to use an organizational account to avoid problems if specific individuals leave your organization. When you set up a Google social identity provider, you'll need to perform the following tasks:

Plan ahead. It may take some time before the `Google+` API that you configure for IDM is ready for use.

- In the Google API Manager, select and enable the `Google+` API. It is one of the Google "social" APIs.
- Create a project for IDM.
- Create OAuth client ID credentials. You'll need to configure an `OAuth consent screen` with at least a product name and email address.
- When you set up a Web application for the client ID, you'll need to set up a web client with:
  - `Authorized JavaScript origins`

    The origin URL for IDM, typically a URL such as `https://openidm.example.com:8443`
  - `Authorized redirect URIs`

    The redirect URI after users are authenticated, typically, `https://openidm.example.com:8443/`
- In the list of credentials, you'll see a unique `Client ID` and `Client secret`. You'll need this information when you configure the Google social identity provider, as described in "Configure a Google Social Identity Provider".

For Google's procedure, see the Google Identity Platform documentation on *Setting Up OAuth 2.0*.

Configure a Google Social Identity Provider

1. To configure a Google social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Google social identity provider, and if needed, select the edit icon.

3. Include the Google values for `Client ID` and `Client Secret` for your project, as described earlier in this section.
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Google Social Identity Provider Configuration Details".

When you enable a Google social identity provider in the Admin UI, IDM generates the `identityProvider-google.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information from what you configured in the Admin UI, and beyond. The first part of the file includes the name of the provider, endpoints, as well as the values for `clientId` and `clientSecret`.

```
{
    "enabled" : true,
    "authorizationEndpoint" : "https://accounts.google.com/o/oauth2/v2/auth",
    "tokenEndpoint" : "https://www.googleapis.com/oauth2/v4/token",
    "userInfoEndpoint" : "https://www.googleapis.com/oauth2/v3/userinfo",
    "wellKnownEndpoint" : "https://accounts.google.com/.well-known/openid-configuration",
    "issuer": "https://accounts.google.com",
    "clientId" : "<someUUID>",
    "clientSecret" : {encrypted-client-secret},
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, `scopes`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Google social identity, you can verify this by selecting Manage > Google, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Google Social Identity Provider Configuration Details".

Configure User Registration to Link to Google

Once you've configured the Google social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and under the Social tab, enable the option associated with Social Registration. For more information on user self-service features, see "Self-Service End User UI".

When you enable social registration, you're allowing users to register on IDM through all active social identity providers.

Google Social Identity Provider Configuration Details

You can set up the Google social identity provider through the Admin UI or in a `conf/identityProvider-google.json` file. IDM generates the `identityProvider-google.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Google Provider pop-up window, along with associated information in the `identityProvider-google.json` file:

Google Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Google Identity Platform project.
Client Secret	`clientSecret`	Used with the Client ID to access the configured Google API.
Scope	`scope`	An array of strings that allows access to user data; see Google's documentation on Authorization Scopes.
Authorization Endpoint	`authorizationEndpoint`	Per *RFC 6749*, "used to interact with the resource owner and obtain an authorization grant". For Google's implementation, see *Forming the URL*.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization grant, and returns an access and ID token.
User Info Endpoint	`userInfoEndpoint`	Endpoint that receives an access token, and returns information about the user.
Well-Known Endpoint	`wellKnownEndpoint`	Access URL for Google's *Discovery Document*.
Issuer	`issuer`	The token issuer. Typically, `https://accounts.google.com`.
Not in the Admin UI	`name`	Name of the social identity provider.
Not in the Admin UI	`type`	Authentication module.
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider.
Not in the Admin UI	`propertyMap`	Mapping between Google and IDM.

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Instagram Social Identity Provider

1. "Set Up Instagram"
2. "Configure an Instagram Social Identity Provider"

3. "Configure User Registration to Link to Instagram"
4. "Instagram Social Identity Provider Configuration Details"

Set Up Instagram

To set up Instagram as a social identity provider, navigate to *Facebook for Developers*, and follow the steps. You'll need a minimum of:

- An Instagram account
- A Facebook developer account
- An application name and description
- A website URL for your app, such as `http://openidm.example.com:8080`
- A Redirect URL for IDM, such as `http://openidm.example.com:8080/`

Configure an Instagram Social Identity Provider

To Configure an Instagram Social Identity Provider:

1. Log in to the Admin UI.
2. From the navigation bar, click Configure > Social ID Providers.
3. Enable the Instagram social identity provider.
4. In the `Instagram Provider` modal, verify the Redirect URI matches what you entered in "Set Up Instagram".
5. Enter the Client ID and Client Secret from "Set Up Instagram".
6. Enter the other configuration details and `Advanced Options`, per the following:
   - "Instagram Social Identity Provider Configuration Details"
   - "Social Identity Provider Button and Badge Properties"
7. Click Save.

When you enable an Instagram social identity provider in the Admin UI, IDM generates a `conf/identityProvider-instagram.json` file. The file contains all options configured using the Admin UI and more. The first part of the file includes the name of the provider, endpoints, and scopes, as well as the values of `clientId` and `clientSecret`.

```
{
   "provider" : "instagram",
   ...
   "clientId" : "<Client_ID_Name>",
   "clientSecret" : {
      "$crypto" : {
         "type" : "x-simple-encryption",
         "value" : {
            "cipher" : "AES/CBC/PKCS5Padding",
            "stableId" : "openidm-sym-default",
            "salt" : "<hashValue>",
            "data" : "<encryptedValue>",
            "keySize" : 16,
            "purpose" : "idm.config.encryption",
            "iv" : "<encryptedValue>",
            "mac" : "<hashValue>"
         }
      }
   },
   "authorizationEndpoint" : "https://api.instagram.com/oauth/authorize/",
   "tokenEndpoint" : "https://api.instagram.com/oauth/access_token",
   "userInfoEndpoint" : "https://graph.instagram.com/me?fields=id,username",
   "redirectUri" : "http://openidm.example.com:8080/",
   "scope" : [
       "user_profile",
   ],
...
```

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

The file includes `schema` information for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with an Instagram social identity, you can verify this by selecting Manage > Instagram, and then selecting a user. For more information about the properties in this file, refer to "Instagram Social Identity Provider Configuration Details".

Configure User Registration to Link to Instagram

After you configure the Instagram social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

Note

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Instagram Social Identity Provider Configuration Details

You can set up the Instagram social identity provider through the Admin UI or in a `conf/identityProvider-instagram.json` file. IDM generates the `identityProvider-instagram.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Instagram Provider pop-up window, along with associated information in the `identityProvider-instagram.json` file:

Instagram Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	Your Instagram App client identifier
Client Secret	`clientSecret`	Used with the Client ID to access the Instagram API
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	Typically `https://api.instagram.com/oauth/authorize/`; known as an Instagram *Authorize URL*
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.instagram.com/oauth/access_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://graph.instagram.com/me?fields=id,username`
Not in the Admin UI	`provider`	Name of the social identity provider
Not in the Admin UI	`configClass`	Configuration class for the authentication module
Not in the Admin UI	`basicAuth`	Whether to use basic authentication
Not in the Admin UI	`propertyMap`	Mapping between Instagram and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

LinkedIn Social Identity Provider

- "Set Up a LinkedIn App"
- "Configure a LinkedIn Social Identity Provider"

- "Configure User Registration With LinkedIn"
- "LinkedIn Social Identity Provider Configuration Details"

Set Up a LinkedIn App

Before you start, you will need a LinkedIn account. You can use a personal LinkedIn account for testing, but you should ultimately use an organizational account to avoid problems if individuals leave your organization.

To set up a LinkedIn app:

1. Log in to LinkedIn and navigate to LinkedIn Developers -> MyApps.
2. Select Create app and enter the following information:
   - **App name.** Enter any unique name that is fewer than 50 characters.
   - **Company.** The company name that will be associated with this application.
   - **Privacy policy URL.** An optional URL that displays your company's Privacy Policy.
   - **Business email.** The business email address that is associated with this application.
   - **App logo.** The logo that is displayed to users when they authenticate with this app.
3. Select the products that should be integrated into the app.
4. Accept LinkedIn's legal terms.
5. Select Verify to associate the app with your company, then follow the verification approval process.
6. After you have approved the app, select it under My Apps, then select the Auth tab.
7. Take note of the `Client ID` and `Client Secret`—you will need them in the next procedure.
8. The app should have the following Permissions:
   - `r_emailaddress`
   - `r_liteprofile`
   - `w_member_social`
9. Under OAuth 2.0 settings, select Add redirect URL and enter the FQDN and port number of your IDM instance. For example, `http://openidm.example.com:8080/`

Note

For LinkedIn's procedure, see their documentation on Authenticating with OAuth 2.0.

Configure a LinkedIn Social Identity Provider

1. To configure a LinkedIn social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the LinkedIn social identity provider.
3. Make sure that the Redirect URI on this screen matches the OAuth 2.0 Redirect URL that you entered in "Set Up a LinkedIn App".
4. Copy the `Client ID` and `Client Secret` that you obtained in "Set Up a LinkedIn App".
5. (Optional) Change any of the `Advanced Options` listed in "LinkedIn Social Identity Provider Configuration Details".
6. Select Save.

When you enable a LinkedIn social identity provider, IDM generates the corresponding `identityProvider-linkedIn.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, endpoints, as well as the `clientId` and encrypted `clientSecret`.

```
{
    "provider" : "linkedIn",
    "authorizationEndpoint" : "https://www.linkedin.com/oauth/v2/authorization",
    "tokenEndpoint" : "https://www.linkedin.com/oauth/v2/accessToken",
    "userInfoEndpoint" : "https://api.linkedin.com/v2/me?
projection=(id,firstName,lastName,profilePicture(displayImage~:playableStreams))",
    "emailAddressEndpoint" : "https://api.linkedin.com/v2/emailAddress?
q=members&projection=(elements*(handle~))",
    "clientId" : "77l9udb8qmqihq",
    "clientSecret" : {
        "$crypto" : {
            "type" : "x-simple-encryption",
            "value" : {
                "cipher" : "AES/CBC/PKCS5Padding",
                "stableId" : "openidm-sym-default",
                "salt" : "2cmC36Ds++6xAtRhlvNOEw==",
                "data" : "TJ7VOHjJI0VWWedTKX4agviqc3H3Un5RDVAWyB2u64g=",
                "keySize" : 16,
                "purpose" : "idm.config.encryption",
                "iv" : "QbGAUSuOMrCh1i8F0fWGyA==",
                "mac" : "rUFVcSJ5+s+LZL6YFB3rFQ=="
            }
        }
    },
    "scope" : [
        "r_liteprofile",
        "r_emailaddress"
    ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

The file includes `schema` information, indicating the properties of each social identity account that will be collected by IDM, and the order in which these properties appear in the Admin UI. When you have registered a user with a LinkedIn social identity, you can verify these properties by selecting Manage > LinkedIn, then selecting the user.

Further down in the file, the `propertyMap` maps user information between the `source` (social identity provider) and the `target` (IDM).

For more information about the properties in this file, see "LinkedIn Social Identity Provider Configuration Details".

Configure User Registration With LinkedIn

After you have configured the LinkedIn social identity provider, activate it by enabling User Registration:

1. Select Configure > User Registration > Enable.
2. On the Social tab, enable Social Registration. For more information about user self-service features, see "Self-Service End User UI".

Note

When you enable social registration, you are allowing users to register in IDM through all active social identity providers.

LinkedIn Social Identity Provider Configuration Details

You can set up the LinkedIn social identity provider through the Admin UI or in a `conf/identityProvider-linkedIn.json` file. IDM generates the `identityProvider-linkedIn.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI LinkedIn Provider pop-up window, along with associated information in the `identityProvider-linkedIn.json` file:

LinkedIn Social Identity Provider Configuration Properties

Property (UI)	**Property (JSON file)**	**Description**
Client ID	`clientId`	The client identifier for your LinkedIn Application
Client Secret	`clientSecret`	Used with the Client ID to access the applicable LinkedIn API
Scope	`scope`	An array of strings that allows access to user data; see LinkedIn's documentation on *Lite Profile Fields*.
Authorization Endpoint	`authorizationEndpoint`	Per *RFC 6749*, "used to interact with the resource owner and obtain an authorization grant". For LinkedIn's implementation, see their documentation on *Authenticating with OAuth 2.0*.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. For LinkedIn's implementation, see their documentation on *Authenticating with OAuth 2.0*.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields through LinkedIn's API.
Email Address Endpoint	`emailAddressEndpoint`	API that must be called to retrieve the email address of the user.
Well-Known Endpoint	`wellKnownEndpoint`	Not used for LinkedIn
Not in the Admin UI	`name`	Name of the social identity provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between LinkedIn and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Microsoft Social Identity Provider

- "Set Up Microsoft"
- "Configure a Microsoft Social Identity Provider"
- "Configure User Registration to Link to Microsoft"
- "Microsoft Social Identity Provider Configuration Details"

Note

Microsoft as a social identity provider requires access over secure HTTP (HTTPS). This example assumes that you've configured IDM on `https://openidm.example.com:8443`. Substitute your URL for `openidm.example.com`.

Set Up Microsoft

For Microsoft documentation on how to set up a social identity provider, navigate to the following article: *Sign-in Microsoft Account & Azure AD users in a single app* . You'll need a Microsoft account.

To set up Microsoft as a social identity provider:

- Navigate to the Microsoft app registration portal at https://apps.dev.microsoft.com/ and sign in with your Microsoft account.
- Select *Add an App* and give your app a name.

  The portal will assign your app a unique `Application ID`.
- To find your `Application Secret`, select *Generate New Password*. That password is your Application Secret.

  Tip

  Save your new password. It is the only time you'll see the `Application Secret` for your new app.
- Select Add Platform. You'll choose a `Web` platform, enable `Allow Implicit Flow` and set up the following value for `Redirect URI`:
  - `https://openidm.example.com:8443/`

If desired, you can also enter the following information:

- Logo image

- Terms of Service URL
- Privacy Statement URL

The `OAuth2` credentials for your new Microsoft App include an `Application ID` and `Application Secret` for your app.

Configure a Microsoft Social Identity Provider

1. To configure a Microsoft social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Microsoft social identity provider.

   In the `Microsoft Provider` pop-up that appears, the values for `Redirect URI` should match the values that you've entered for Allowed Return URLs in "Set Up Microsoft".
3. Include the values that Microsoft created for `Client ID` and `Client Secret`, as described in "Set Up Microsoft".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Microsoft Social Identity Provider Configuration Details".

When you enable a Microsoft social identity provider in the Admin UI, IDM generates the `identityProvider-microsoft.json` file in your project's `conf/` subdirectory.

It includes parts of the file that you may have configured through the Admin UI. While the labels in the UI specify Application ID and Application Secret, you'll see them as `clientId` and `clientSecret`, respectively, in the configuration file.

```
"provider" : "microsoft",
   "authorizationEndpoint" : "https://login.microsoftonline.com/common/oauth2/v2.0/authorize",
   "tokenEndpoint" : "https://login.microsoftonline.com/common/oauth2/v2.0/token",
   "userInfoEndpoint" : "https://graph.microsoft.com/v1.0/me"
   "clientId" : "<someUUID>",
   "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
   },
   "scope" : [
       "User.Read"
   ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Microsoft social identity, you can verify this by selecting Manage > Microsoft, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Microsoft Social Identity Provider Configuration Details".

Configure User Registration to Link to Microsoft

Once you've configured the Microsoft social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Microsoft Social Identity Provider Configuration Details

You can set up the Microsoft social identity provider through the Admin UI or in a `conf/identityProvider-microsoft.json` file. IDM generates the `identityProvider-microsoft.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Microsoft Provider pop-up window, along with associated information in the `identityProvider-microsoft.json` file:

Microsoft Social Identity Provider Configuration Properties

Property (UI)	**Property (JSON file)**	**Description**
Application ID	`clientId`	The client identifier for your Microsoft App
Application Secret	`clientSecret`	Used with the Application ID; shown as application password
Scope	`scope`	OAuth 2 scopes; for more information, see *Microsoft Graph Permission Scopes*.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://login.microsoftonline.com/common/oauth2/v2.0/authorize`

Property (UI)	Property (JSON file)	Description
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code and returns an access token; typically `https://login.microsoftonline.com/common/oauth2/v2.0/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://graph.microsoft.com/v1.0/me`
Not in the Admin UI	`name`	Name of the social identity provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between Microsoft and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Salesforce Social Identity Provider

- "Set Up Salesforce"
- "Configure a Salesforce Social Identity Provider"
- "Configure User Registration to Link to Salesforce"
- "Salesforce Social Identity Provider Configuration Details"

Note

When you configure a Salesforce app, look for a *Consumer Key* and a *Consumer Secret*. IDM uses this information as a `clientId` and `clientSecret`, respectively.

For reference, read through the following Salesforce documentation: *Connected Apps Overview*.

Set Up Salesforce

To set up Salesforce as a social identity provider, you will need a Salesforce developer account. Log in to the *Salesforce Developers Page* with your developer account credentials and create a new Connected App.

Note

These instructions were written with the Winter '19 Release of the Salesforce API. The menu items might differ slightly if you are working with a different version of the API.

Under App Setup, select Create > Apps > Connected Apps > New. You will need to add the following information:

- Connected App Name
- API Name (defaults to the Connected App Name)
- Contact Email
- Activate the following option: *Enable OAuth Settings*
- Callback URL (also known as the *Redirect URI* for other providers), for example `https://localhost:8443/`.

  The Callback URL must correspond to the URL that you use to log in to the IDM Admin UI.
- Add the following OAuth scopes:
  - Access and Manage your data (api)
  - Access your basic information (id, profile, email, address, phone)
  - Perform requests on your behalf at any time (refresh_token, offline_access)
  - Provide access to your data via the Web (web)

  Note that these must be added even if you are otherwise planning to use the `full` OAuth scope.

After you have saved the Connected App, it might take a few minutes for the new app to appear under Administration Setup > Manage Apps > Connected Apps.

Select the new Connected App then locate the *Consumer Key* and *Consumer Secret* (under the API list). You'll use that information as shown here:

- Salesforce Consumer Key = IDM Client ID
- Salesforce Consumer Secret = IDM Client Secret

Configure a Salesforce Social Identity Provider

1. To configure a Salesforce social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Salesforce social identity provider.

   In the `Salesforce Provider` pop-up that appears, the values for `Redirect URI` should match the value that you've entered for Callback URL in "Set Up Salesforce".
3. Include the values that Salesforce created for `Consumer Key` and `Consumer Secret`, as described in "Set Up Salesforce".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Salesforce Social Identity Provider Configuration Details".

When you enable a Salesforce social identity provider in the Admin UI, IDM generates the `identityProvider-salesforce.json` file in your project's `conf/` subdirectory.

It includes parts of the file that you may have configured through the Admin UI. While the labels in the UI specify Consumer Key and Consumer Secret, you'll see them as `clientId` and `clientSecret`, respectively, in the configuration file.

```
{
    "provider" : "salesforce",
    "authorizationEndpoint" : "https://login.salesforce.com/services/oauth2/authorize",
    "tokenEndpoint" : "https://login.salesforce.com/services/oauth2/token",
    "userInfoEndpoint" : "https://login.salesforce.com/services/oauth2/userinfo",
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
    },
    "scope" : [
        "id",
        "api",
        "web"
    ],
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Salesforce social identity, you can verify this by selecting Manage > Salesforce, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Salesforce Social Identity Provider Configuration Details".

Configure User Registration to Link to Salesforce

Once you've configured the Salesforce social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Salesforce Social Identity Provider Configuration Details

You can set up the Salesforce social identity provider through the Admin UI or in a `conf/identityProvider-salesforce.json` file. IDM generates the `identityProvider-salesforce.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Salesforce Provider pop-up window, along with associated information in the `identityProvider-salesforce.json` file:

Salesforce Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Salesforce App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Salesforce API
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	A typical URL: `https://login.salesforce.com/services/oauth2/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; such as `https://login.salesforce.com/services/oauth2/token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; a typical URL: `https://login.salesforce.com/services/oauth2/userinfo`
Not in the Admin UI	`provider`	Name of the social identity provider
Not in the Admin UI	`configClass`	Configuration class for the authentication module
Not in the Admin UI	`basicAuth`	Whether to use basic authentication
Not in the Admin UI	`propertyMap`	Mapping between Salesforce and IDM

Twitter Social Identity Provider

- "Set Up Twitter"
- "Configure Twitter as a Social Identity Provider"
- "Configure User Registration to Link to Twitter"

- "Twitter Social Identity Provider Configuration Details"

Set Up Twitter

To set up Twitter as a social identity provider, navigate to the following page: *Single-user OAuth with Examples* . You'll need a Twitter account. You can then navigate to the Twitter *Application Management* page, where you can select *Create New App* and enter at least the following information:

- Name
- Description
- Website, such as `http://openidm.example.com:8080`
- Callback URL, such as `http://openidm.example.com:8080/`; required for IDM; for other providers, known as `RedirectURI`

When complete and saved, you should see a `Consumer Key` and `Consumer Secret` for your new web app.

Note

Twitter Apps use the OAuth 1.0a protocol. Fortunately, with IDM, you can use the same process used to configure OIDC and OAuth 2 social identity providers.

Configure Twitter as a Social Identity Provider

1. To configure Twitter as a social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Twitter social identity provider.

   In the `Twitter Provider` pop-up that appears, the values for `Callback URL` should use the same value shown in "Set Up Twitter".
3. Include the values that Twitter created for `Consumer Key` and `Consumer Secret`, as described in "Set Up Twitter".
4. Under regular and `Advanced Options`, if necessary, include the options shown in the following appendix: "Twitter Social Identity Provider Configuration Details".

When you enable a Twitter social identity provider in the Admin UI, IDM generates the `identityProvider-twitter.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, endpoints, as well as information from the

Consumer Key and *Consumer Secret*, you'll see them as `clientId` and `clientSecret`, respectively, in the configuration file.

```
{
    "provider" : "twitter",
    "requestTokenEndpoint" : "https://api.twitter.com/oauth/request_token",
    "authorizationEndpoint" : "https://api.twitter.com/oauth/authenticate",
    "tokenEndpoint" : "https://api.twitter.com/oauth/access_token",
    "userInfoEndpoint" : "https://api.twitter.com/1.1/account/verify_credentials.json",
    "clientId" : "<Client_ID_Name>",
    "clientSecret" : {
      "$crypto" : {
          "type" : "x-simple-encryption",
          "value" : {
              "cipher" : "AES/CBC/PKCS5Padding",
              "stableId" : "openidm-sym-default",
              "salt" : "<hashValue>",
              "data" : "<encryptedValue>",
              "keySize" : 16,
              "purpose" : "idm.config.encryption",
              "iv" : "<encryptedValue>",
              "mac" : "<hashValue>"
          }
      }
    },
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`.

The next part of the file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Twitter social identity, you can verify this by selecting Manage > Twitter, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Twitter Social Identity Provider Configuration Details".

Configure User Registration to Link to Twitter

Once you've configured the Twitter social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Twitter Social Identity Provider Configuration Details

You can set up the Twitter social identity provider through the Admin UI or in a `conf/identityProvider-twitter.json` file. IDM generates the `identityProvider-twitter.json` file when you configure and enable the Twitter social identity provider in the Admin UI. Alternatively, you can create that file manually.

The following table includes the information shown in the Admin UI Twitter Provider pop-up window, along with associated information in the `identityProvider-twitter.json` file.

Twitter Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Consumer Key	`clientId`	The client identifier for your Twitter App
Consumer Secret	`clientSecret`	Used with the Client ID to access the applicable Twitter API
Authorization Endpoint	`authorizationEndpoint`	Typically `https://api.twitter.com/oauth/authenticate`; known as a Twitter *Authorize URL*
Access Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.twitter.com/oauth/access_token`
User Info Endpoint	`userInfoEndpoint`	Access for other URIs; typically `https://api.twitter.com/1.1/account/verify_credentials.json`
Request Token Endpoint	`requestTokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.twitter.com/oauth/request_token`
Not in the Admin UI	`provider`	Name of the social identity provider
Not in the Admin UI	`authenticationIdKey`	The user identity property, such as `_id`
Not in the Admin UI	`configClass`	Configuration class for the authentication module
Not in the Admin UI	`basicAuth`	Whether to use basic authentication
Not in the Admin UI	`propertyMap`	Mapping between Twitter and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Setting Up Vkontakte as an IDM Social Identity Provider

- "Set Up Vkontakte"
- "Configure a Vkontakte Social Identity Provider"
- "Configure User Registration to Link to Vkontakte"
- "Vkontakte Social Identity Provider Configuration Details"

Note

When you configure a Vkontakte app, look for an *Application ID* and a *Secure Key*. IDM uses this information as a `clientId` and `clientSecret`, respectively.

Set Up Vkontakte

To set up Vkontakte as a social identity provider, navigate to the following Vkontakte page: *Vkontakte Developers Page* . You'll need a Vkontakte account. Find a *My Apps* link. You can then create an application with the following information:

- Title (The name of your app)
- Platform (Choose Website)
- Site Address (The URL of your IDM deployment, such as `http://openidm.example.com:8080/`
- Base domain (Example: `example.com`)
- Authorized Redirect URI (Example: `http://openidm.example.com:8080/`)
- API Version; for the current VKontakte API version, see *VK Developers Documentation, API Versions* section. The default VKontakte API version used for IDM 7.1 is 5.73.

If you leave and need to return to Vkontakte, navigate to `https://vk.com/dev` and select *My Apps*. You can then *Manage* the new apps that you've created.

Navigate to the *Settings* for your app, where you'll find the *Application ID* and *Secure Key* for your app. You'll use that information as shown here:

- Vkontakte Application ID = IDM Client ID
- Vkontakte Secure Key = IDM Client Secret

Configure a Vkontakte Social Identity Provider

1. To configure a Vkontakte social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Vkontakte social identity provider.

   In the `Vkontakte Provider` pop-up that appears, the values for `Redirect URI` should match the values that you've entered for Authorized Redirect URI in "Set Up Vkontakte".
3. Include the values that Vkontakte created for `Client ID` and `Client Secret`, as described in "Set Up Vkontakte".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "Vkontakte Social Identity Provider Configuration Details".

When you enable a Vkontakte social identity provider in the Admin UI, IDM generates the `identityProvider-vkontakte.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, endpoints, as well as information from the *Consumer Key* and *Consumer Secret*, you'll see them as `clientId` and `clientSecret`, respectively, in the configuration file.

```
{
    "provider" : "vkontakte",
    "configClass" : "org.forgerock.oauth.clients.vk.VKClientConfiguration",
    "basicAuth" : false,
    "clientId" : "<someUUID>",
    "clientSecret" : {
        "$crypto" : {
            "type" : "x-simple-encryption",
            "value" : {
                "cipher" : "AES/CBC/PKCS5Padding",
                "stableId" : "openidm-sym-default",
                "salt" : "<hashValue>",
                "data" : "<encryptedValue>",
                "keySize" : 16,
                "purpose" : "idm.config.encryption",
                "iv" : "<encryptedValue>",
                "mac" : "<hashValue>"
            }
        }
    },
    "authorizationEndpoint" : "https://oauth.vk.com/authorize",
    "tokenEndpoint" : "https://oauth.vk.com/access_token",
    "userInfoEndpoint" : "https://api.vk.com/method/users.get",
    "redirectUri" : "http://openidm.example.com:8080/",
    "apiVersion" : "5.73",
    "scope" : [
        "email"
    ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Vkontakte social identity, you can verify this by selecting Manage > Vkontakte, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "Vkontakte Social Identity Provider Configuration Details".

Configure User Registration to Link to Vkontakte

Once you've configured the Vkontakte social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Vkontakte Social Identity Provider Configuration Details

You can set up the Vkontakte social identity provider through the Admin UI or in a `conf/identityProvider-vkontakte.json` file. IDM generates the `identityProvider-vkontakte.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Vkontakte Provider pop-up window, along with associated information in the `identityProvider-vkontakte.json` file:

Vkontakte Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Application ID	`clientId`	The client identifier for your Vkontakte App
Secure Key	`clientSecret`	Used with the Client ID to access the applicable Vkontakte API
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically `https://oauth.vk.com/authorize`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `"https://oauth.vk.com/access_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.vk.com/method/users.get`
API Version	`apiVersion`	Version of the applicable VKontakte API, available from *VK Developers Documentation, API Versions* section. The default VKontakte API version used for IDM 7.1 is 5.73.
Not in the Admin UI	`provider`	Name of the social identity provider
Not in the Admin UI	`configClass`	Configuration class for the authentication module
Not in the Admin UI	`basicAuth`	Whether to use basic authentication
Not in the Admin UI	`authenticationIdKey`	The user identity property, such as `id`
Not in the Admin UI	`propertyMap`	Mapping between Vkontakte and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

WeChat Social Identity Provider

- "Set Up WeChat"
- "Configure a WeChat Social Identity Provider"
- "Configure User Registration to Link to WeChat"
- "WeChat Social Identity Provider Configuration Details"

These requirements assume that you have a WeChat developer account where you can get access to create WeChat web application credentials. To verify access, you'll need the WeChat app on your mobile phone.

Set Up WeChat

To set up WeChat as a social identity provider, you'll need to get the following information for your WeChat app. The name may be different in WeChat.

- Client ID (WeChat uses `appid` as of this writing.)
- Client Secret (WeChat uses `secret` as of this writing.)
- Scope
- Authorization Endpoint URL
- Token Endpoint URL
- User Info Endpoint URL
- Redirect URI, normally something like `http://openidm.example.com/`

WeChat Unique Requirements

Before testing WeChat, be prepared for the following special requirements:

- WeChat works only if you deploy IDM on one of the following ports: *80* or *443*.

  For more information on how to configure IDM to use these ports, see "*Host and Port Information*" in the *Installation Guide*.
- For registration and sign-in, WeChat requires the use of a mobile device with a Quick Response (QR) code reader.

- For sign-in, you'll also need to install the WeChat app on your mobile device.

Configure a WeChat Social Identity Provider

1. To configure a WeChat social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the WeChat social identity provider.

   In the `WeChat Provider` pop-up that appears, the values for `Redirect URI` should match the values that you've entered for Allowed Return URLs in "Set Up WeChat".
3. Include the values that WeChat created for `Client ID` and `Client Secret`, as described in "Set Up WeChat".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "WeChat Social Identity Provider Configuration Details".

When you enable a WeChat social identity provider in the Admin UI, IDM generates the `identityProvider-wechat.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information from what you configured in the Admin UI, and beyond. The first part of the file includes the name of the provider, endpoints, scopes, as well as the values for `clientId` and `clientSecret`.

```
{
    "provider" : "wechat",
    ...
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
              "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
    },
    "authorizationEndpoint" : "https://open.weixin.qq.com/connect/qrconnect",
    "tokenEndpoint" : "https://api.wechat.com/sns/oauth2/access_token",
    "refreshTokenEndpoint" : "https://api.wechat.com/sns/oauth2/refresh_token",
    "userInfoEndpoint" : "https://api.wechat.com/sns/userinfo",
    "redirectUri" : "http://openidm.example.com:8080/",
    "scope" : [
        "snsapi_login"
    ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a WeChat social identity, you can verify this by selecting Manage > WeChat, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "WeChat Social Identity Provider Configuration Details".

Configure User Registration to Link to WeChat

Once you've configured the WeChat social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

WeChat Social Identity Provider Configuration Details

You can set up the WeChat social identity provider through the Admin UI or in a `conf/identityProvider-wechat.json` file. IDM generates the `identityProvider-wechat.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI WeChat Provider pop-up window, along with associated information in the `identityProvider-wechat.json` file.

Note

WeChat supports URLs on one of the following ports: 80 or 443. For more information on how to configure IDM to use these ports, see "*Host and Port Information*" in the *Installation Guide*.

WeChat Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your WeChat App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable WeChat API

Property (UI)	Property (JSON file)	Description
Scope	`scope`	An array of strings that allows access to user data
Authorization Endpoint	`authorizationEndpoint`	Typically `https://open.weixin.qq.com/connect/qrconnect`.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://api.wechat.com/sns/oauth2/access_token`
Refresh Token Endpoint	`refreshTokenEndpoint`	Endpoint that receives a one-time authorization code, and returns a refresh token; typically `https://api.wechat.com/sns/oauth2/refresh_token`
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://api.wechat.com/user/profile`
Not in the Admin UI	`provider`	Name of the social identity provider
Not in the Admin UI	`configClass`	Configuration class for the authentication module
Not in the Admin UI	`basicAuth`	Whether to use basic authentication
Not in the Admin UI	`propertyMap`	Mapping between WeChat and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

WordPress Social Identity Provider

- "Set Up WordPress"
- "Configure a WordPress Social Identity Provider"
- "Configure User Registration to Link to WordPress"
- "WordPress Social Identity Provider Configuration Details"

Set Up WordPress

To set up WordPress as a social identity provider, navigate to the following WordPress Developers page: *Developer Resources* . You'll need a WordPress account. You can then navigate to the WordPress *My Applications* page, where you can create a new web application, with the following information:

- Name
- Description
- Website URL, which becomes your Application URL
- Redirect URL(s); for IDM, normally `http://openidm.example.com:8080/`

- Type, which allows you to select Web clients

When complete and saved, you should see a list of `OAuth Information` for your new web application. That information should include your `Client ID` and `Client Secret`.

Configure a WordPress Social Identity Provider

1. To configure a WordPress social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the WordPress social identity provider.

   In the `WordPress Provider` pop-up that appears, the values for `Redirect URI` should match the values that you've entered for Allowed Return URLs in "Set Up WordPress".
3. Include the values that WordPress created for `Client ID` and `Client Secret`, as described in "Set Up WordPress".
4. Under regular and `Advanced Options`, include the options shown in the following appendix: "WordPress Social Identity Provider Configuration Details".

When you enable a WordPress social identity provider in the Admin UI, IDM generates the `identityProvider-wordpress.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, endpoints, as well as the values for `clientId` and `clientSecret`.

```
{
    "provider" : "wordpress",
    "authorizationEndpoint" : "https://public-api.wordpress.com/oauth2/authorize",
    "tokenEndpoint" : "https://public-api.wordpress.com/oauth2/token",
    "userInfoEndpoint" : "https://public-api.wordpress.com/rest/v1.1/me/",
    "enabled" : true,
    "clientId" : "<someUUID>",
    "clientSecret" : {
       "$crypto" : {
           "type" : "x-simple-encryption",
           "value" : {
               "cipher" : "AES/CBC/PKCS5Padding",
               "stableId" : "openidm-sym-default",
               "salt" : "<hashValue>",
               "data" : "<encryptedValue>",
               "keySize" : 16,
               "purpose" : "idm.config.encryption",
               "iv" : "<encryptedValue>",
               "mac" : "<hashValue>"
           }
       }
    },
    "scope" : [
        "auth"
    ],
...
```

You should also see UI settings related to the social identity provider icon (badge) and the sign-in button, described in "Social Identity Provider Button and Badge Properties".

You'll see links related to the `authenticationIdKey`, `redirectUri`, and `configClass`; the location may vary.

The file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Wordpress social identity, you can verify this by selecting Manage > Wordpress, and then selecting a user.

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

If you need more information about the properties in this file, refer to the following appendix: "WordPress Social Identity Provider Configuration Details".

Configure User Registration to Link to WordPress

Once you've configured the WordPress social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

WordPress Social Identity Provider Configuration Details

You can set up the WordPress social identity provider through the Admin UI or in a `conf/identityProvider-wordpress.json` file. IDM generates the `identityProvider-wordpress.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI WordPress Provider pop-up window, along with associated information in the `identityProvider-wordpress.json` file:

WordPress Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your WordPress App
Client Secret	`clientSecret`	Used with the Client ID to access the applicable WordPress API
Scope	`scope`	An array of strings that allows access to user data; see WordPress's *OAuth2 Authentication* Documentation.

Property (UI)	Property (JSON file)	Description
Authorization Endpoint	`authorizationEndpoint`	Typically `https://public-api.wordpress.com/oauth2/authorize`; known as a WordPress *Authorize URL*.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token; typically `https://public-api.wordpress.com/oauth2/token`; known as a WordPress *Request Token URL*.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields; typically `https://public-api.wordpress.com/rest/v1.1/me/`
Not in the Admin UI	`name`	Name of the social identity provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between WordPress and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Yahoo Social Identity Provider

- "Set Up Yahoo"
- "Configure Yahoo as a Social Identity Provider"
- "Configure User Registration to Link to Yahoo"
- "Yahoo Social Identity Provider Configuration Details"

Set Up Yahoo

To set up Yahoo as a social identity provider, navigate to the following page: *Yahoo OAuth 2.0 Guide* . You'll need a Yahoo account. You can then navigate to the *Create an App* page, where you can follow the Yahoo process to create a new web application with the following information:

- Application Name
- Web Application
- Callback Domain, such as `openidm.example.com`; required for IDM
- API Permissions; for whatever you select, choose *Read/Write*. IDM only reads Yahoo user information.

When complete and saved, you should see a `Client ID` and `Client Secret` for your new web app.

Note

Yahoo supports URLs using only HTTPS, only on port 443. For more information on how to configure IDM to use these ports, see "*Host and Port Information*" in the *Installation Guide*.

Configure Yahoo as a Social Identity Provider

1. To configure Yahoo as a social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the Yahoo social identity provider.

   In the `Yahoo Provider` pop-up that appears, the values for `Redirect URI` should use the same *Callback Domain* as shown in "Set Up Yahoo".
3. Include the values that Yahoo created for `Client ID` and `Client Secret`, as described in "Set Up Yahoo".
4. Click Show Advanced Options, and enter information, as necessary. A complete list of options is available here: "Yahoo Social Identity Provider Configuration Details".

When you enable a Yahoo social identity provider in the Admin UI, IDM generates the `identityProvider-yahoo.json` file in your project's `conf/` subdirectory.

When you review that file, you should see information beyond what you see in the Admin UI. The first part of the file includes the name of the provider, the scope, and UI settings related to the social identity provider icon (badge) and the sign-in button. For more information on the icon and button, see "Social Identity Provider Button and Badge Properties".

```
{
    "provider" : "yahoo",
    "scope" : [
        "openid",
        "sdpp-w"
    ],
    "uiConfig" : {
        "iconBackground" : "#7B0099",
        "iconClass" : "fa-yahoo",
        "iconFontColor" : "white",
        "buttonClass" : "fa-yahoo",
        "buttonDisplayName" : "Yahoo",
        "buttonCustomStyle" : "background-color: #7B0099; border-color: #7B0099; color:white;",
        "buttonCustomStyleHover" : "background-color: #7B0099; border-color: #7B0099; color:white;"
    },
```

Another part of the file includes a `propertyMap`, which maps user information entries between the `source` (social identity provider) and the `target` (IDM).

The next part of the file includes `schema` information, which includes properties for each social identity account, as collected by IDM, as well as the order in which it appears in the Admin UI. When you've registered a user with a Yahoo social identity, you can verify this by selecting Manage > Yahoo, and then selecting a user.

Next, there's the part of the file that you may have configured through the Admin UI, plus additional information on the `redirectUri`, the `configClass`, and the `authenticationIdKey`:

```
    "authorizationEndpoint" : "https://api.login.yahoo.com/oauth2/request_auth",
    "tokenEndpoint" : "https://api.login.yahoo.com/oauth2/get_token",
    "wellKnownEndpoint" : "https://api.login.yahoo.com/.well-known/openid-configuration",
    "issuer" : "https://api.login.yahoo.com",
    "clientId" : "<Client_ID_Name>",
    "clientSecret" : {encrypted-client-secret},
    "authenticationIdKey" : "sub",
    "redirectUri" : "https://openidm.example.com/",
    "basicAuth" : false,
    "configClass" : "org.forgerock.oauth.clients.oidc.OpenIDConnectClientConfiguration",
    "enabled" : true
```

If you need more information about the properties in this file, refer to the following appendix: "Yahoo Social Identity Provider Configuration Details".

Configure User Registration to Link to Yahoo

Once you've configured the Yahoo social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and activate that feature. Under the Social tab that appears, enable Social Registration. For more information on IDM user self-service features, see "Self-Service End User UI".

When you enable Social Registration, you're allowing users to register on IDM through all active social identity providers.

Yahoo Social Identity Provider Configuration Details

You can set up the Yahoo social identity provider through the Admin UI or in a `conf/identityProvider-yahoo.json` file. IDM generates the `identityProvider-yahoo.json` file when you configure and enable this social identity provider in the Admin UI. Alternatively, you can create the file manually.

The following table includes the information shown in the Admin UI Yahoo Provider pop-up window, along with associated information in the `identityProvider-yahoo.json` file.

Note

Yahoo supports URLs using only HTTPS, only on port 443. For more information on how to configure IDM to use these ports, see "*Host and Port Information*" in the *Installation Guide*.

Yahoo Social Identity Provider Configuration Properties

Property (UI)	Property (JSON file)	Description
Client ID	`clientId`	The client identifier for your Yahoo App.
Client Secret	`clientSecret`	Used with the Client ID to access the applicable Yahoo API.

Property (UI)	Property (JSON file)	Description
Scope	`scope`	An array of strings that allows access to user data.
Authorization Endpoint	`authorizationEndpoint`	Typically, `https://api.login.yahoo.com/oauth2/request_auth`; known as a Yahoo Authorize URL.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token. Typically, `https://api.login.yahoo.com/oauth2/get_token`.
Well-Known Endpoint	`wellKnownEndpoint`	Access for other URIs. Typically, `https://login.yahoo.com/.well-known/openid-configuration`.
Issuer	`issuer`	The token issuer. Typically, `https://api.login.yahoo.com`.
Not in the Admin UI	`provider`	Name of the social identity provider.
Not in the Admin UI	`configClass`	Configuration class for the authentication module.
Not in the Admin UI	`basicAuth`	Whether to use basic authentication.
Not in the Admin UI	`propertyMap`	Mapping between Yahoo and IDM.

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Custom Social Identity Provider

As suggested in the introduction to this chapter, you'll need to take four basic steps to configure a custom social identity provider:

- "Prepare IDM"
- "Set Up a Custom Social Identity Provider"
- "Configure a Custom Social Identity Provider"
- "Configure User Registration to Link to a Custom Provider"
- "Custom Social Identity Provider Configuration Details"

Note

These instructions require the social identity provider to be *fully* compliant with *The OAuth 2.0 Authorization Framework* or the *OpenID Connect* standards.

Prepare IDM

While IDM includes provisions to work with OpenID Connect 1.0 and OAuth 2.0 social identity providers, connections to those providers are not supported, other than those specifically listed in this chapter. If you haven't already, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory.

To set up another social provider, first add a code block to `conf/identityProviders.json`:

+ *Example Code Block*

```
{
    "provider" : "<providerName>",
    "authorizationEndpoint" : "",
    "tokenEndpoint" : "",
    "userInfoEndpoint" : "",
    "wellKnownEndpoint" : "",
    "clientId" : "",
    "clientSecret" : "",
    "uiConfig" : {
         "iconBackground" : "",
         "iconClass" : "",
         "iconFontColor" : "",
         "buttonImage" : "",
         "buttonClass" : "",
         "buttonCustomStyle" : "",
         "buttonCustomStyleHover" : "",
         "buttonDisplayName" : ""
    },
    "scope" : [ ],
    "authenticationIdKey" : "",
    "schema" : {
       "id" : "urn:jsonschema:org:forgerock:openidm:identityProviders:api:<providerName>",
       "viewable" : true,
       "type" : "object",
       "$schema" : "http://json-schema.org/draft-03/schema",
       "properties" : {
          "id" : {
             "title" : "ID",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "name" : {
             "title" : "Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "first_name" : {
             "title" : "First Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "last_name" : {
             "title" : "Last Name",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "email" : {
             "title" : "Email Address",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          },
          "locale" : {
             "title" : "Locale Code",
             "viewable" : true,
             "type" : "string",
             "searchable" : true
          }
       },
       "order" : [
          "id",
          "name",
          "first_name",
          "last_name",
          "email",
          "locale"
       ],
       "required" : [ ]
    },
    "propertyMap" : [
       {
          "source" : "id",
          "target" : "id"
       },
       {
          "source" : "name",
          "target" : "displayName"
       },
       {
          "source" : "first_name",
          "target" : "givenName"
       },
       {
          "source" : "last_name",
          "target" : "familyName"
       },
       {
          "source" : "email",
          "target" : "email"
       },
       {
          "source" : "email",
          "target" : "username"
       },
       {
          "source" : "locale",
          "target" : "locale"
       }
    ],
    "redirectUri" : "http://openidm.example.com:8080/",
    "configClass" : "org.forgerock.oauth.clients.oidc.OpenIDConnectClientConfiguration",
    "basicAuth" : false,
    "enabled" : true
```

```
},
```

Modify this code block for your selected social provider. Some of these properties may appear under other names. For example, some providers specify an `App ID` that you'd include as a `clientId`.

Additional changes may be required, especially depending on how the provider implements the OAuth2 or OpenID Connect standards.

In the `propertyMap` code block, you should substitute the properties from the selected social identity provider for various values of `source`. Make sure to trace the property mapping through `selfservice.propertymap.json` to the Managed User property shown in `managed.json`. For more information on this multi-step mapping, see "Many Social Identity Providers, One Schema".

As shown in "OpenID Connect Authorization Code Flow", user provisioning information goes through the User Info Endpoint. Some providers, such as LinkedIn and Facebook, may require a list of properties with the endpoint. Consult the documentation for your provider for details.

For more information on the `uiConfig` code block, see "Social Identity Provider Button and Badge Properties".

Both files, `identityProviders.json` and `identityProvider-custom.json`, should include the same information for the new `custom` identity provider. For property details, see "Custom Social Identity Provider Configuration Details".

Once you've included information from your selected social identity provider, proceed with the configuration process. You'll use the same basic steps described for other specified social providers.

Set Up a Custom Social Identity Provider

Every social identity provider should be able to provide the information you need to specify properties in the code block shown in "Prepare IDM".

In general, you'll need an `authorizationEndpoint`, a `tokenEndpoint` and a `userInfoEndpoint`. To link to the custom provider, you'll also have to copy the `clientId` and `clientSecret` that you created with that provider. In some cases, you'll get this information in a slightly different format, such as an `App ID` and `App Secret`.

For the `propertyMap`, check the `source` properties. You may need to revise these properties to match those available from your custom provider.

For examples, refer to the specific social identity providers documented in this chapter.

Configure a Custom Social Identity Provider

1. To configure a custom social identity provider, log in to the Admin UI and navigate to Configure > Social ID Providers.
2. Enable the custom social identity provider. The name you see is based on the `name` property in the relevant code block in the `identityProviders.json` file.

3. If you haven't already done so, include the values provided by your social identity provider for the properties shown. For more information, see the following appendix: "Custom Social Identity Provider Configuration Details".

Configure User Registration to Link to a Custom Provider

Once you've configured a custom social identity provider, you can activate it through User Registration. To do so in the Admin UI, select Configure > User Registration, and under the Social tab, enable the option associated with Social Registration. For more information about user self-service features, see "*Admin UI*" in the *Setup Guide*.

When you enable social identity providers, you're allowing users to register on IDM through all active social identity providers.

Custom Social Identity Provider Configuration Details

When you set up a custom social identity provider, starting with "Prepare IDM", you'll see configuration details in your `conf/identityProviders.json` file. The following table includes the information shown in the relevant Admin UI pop-up window.

IDM generates the content of `identityProvider-custom.json` after you configure and enable the custom social identity provider using the Admin UI. Before you can activate this feature in the Admin UI, copy `/path/to/openidm/samples/example-configurations/self-service/identityProviders.json` to your project's `conf/` directory. You can also manually create this file.

Custom Social Identity Provider Configuration Properties

Property (UI)	**Property (JSON file)**	**Description**
Client ID	`clientId`	The client identifier for your social identity provider
Client Secret	`clientSecret`	Used with the Client ID
Scope	`scope`	An array of strings that allows access to user data; varies by provider.
Authorization Endpoint	`authorizationEndpoint`	Every social identity provider should have an authorization endpoint to authenticate end users.
Token Endpoint	`tokenEndpoint`	Endpoint that receives a one-time authorization code, and returns an access token.
User Info Endpoint	`userInfoEndpoint`	Endpoint that transmits scope-related fields.
Not in the Admin UI	`name`	Name of the social identity provider
Not in the Admin UI	`type`	Authentication module
Not in the Admin UI	`authenticationId`	Authentication identifier, as returned from the User Info Endpoint for each social identity provider
Not in the Admin UI	`propertyMap`	Mapping between the social identity provider and IDM

For information on social identity provider buttons and badges, see "Social Identity Provider Button and Badge Properties".

Configure the Social Providers Authentication Module

The `SOCIAL_PROVIDERS` authentication module incorporates the requirements from social identity providers who rely on either the OAuth2 or OpenID Connect standards. The Social Providers authentication module is enabled by default. To configure or disable this module in the Admin UI, select Configure > Authentication, choose the Modules tab, then select Social Providers from the list of modules.

Authentication settings can be configured from the Admin UI, or by making changes directly in the `authentication.json` file for your project. IDM includes the following code block in the default `authentication.json` file:

```
{
   "name" : "SOCIAL_PROVIDERS",
   "properties" : {
       "defaultUserRoles" : [
           "internal/role/openidm-authorized"
       ],
       "augmentSecurityContext" : {
           "type" : "text/javascript",
           "globals" : { },
           "file" : "auth/populateAsManagedUserFromRelationship.js"
       },
       "propertyMapping" : {
           "userRoles" : "authzRoles"
       }
   },
   "enabled" : true
}
```

The authentication properties are described in detail in "*Authentication and Session Module Configuration*" in the *Authentication and Authorization Guide*.

Account Claiming: Links Between Accounts and Social Identity Providers

If your users have one or more social identity providers, they can link them to the same IDM user account. This section assumes that you have configured one or more of the social identity providers described in "*Social Registration*".

Conversely, you should not be able to link more than one IDM account with a single social identity provider account.

When social accounts are associated with an IDM account, IDM creates a managed record, which uses the name of the social identity provider name as the managed object type, and the subject is used as the `_id`. This combination has a unique constraint; if you try to associate a second IDM account with the same social account, IDM detects a conflict, which prevents the association.

The default process uses the email address associated with the account. Once you've configured social identity providers, you can see this filter in the `selfservice-socialUserClaim.json` file:

```
{
    "name" : "socialUserClaim",
    "identityServiceUrl" : "managed/user",
    "claimQueryFilter" : "/mail eq \"{{mail}}\""
},
```

You can modify the `claimQueryFilter` to a different property such as `telephoneNumber`. Make sure that property is:

- Set to "required" in the `managed.json` file; the default list for managed users is shown here:

  ```
  "required" : [
      "userName",
      "givenName",
      "sn",
      "mail"
  ]
  ```

- Unique; for example, if multiple users have the same telephone number, IDM responds with error messages shown in "When Multiple Users have the Same Email Address".

Based on the `claimQueryFilter`, what IDM does depends on the following scenarios:

- "When the Email Address is New"
- "When One User has the Same Email Address"
- "When Multiple Users have the Same Email Address"

When the Email Address is New

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address doesn't exist for any IDM managed user, IDM takes available identifying information, and pre-populates the self-registration screen. If all required information is included, IDM proceeds to other screens, depending on what you've activated in this section: "*Additional Configuration*".

When One User has the Same Email Address

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address exists for *one* IDM managed user, IDM gives you a chance to link to that account, with the following message:

```
We found an existing account with the same email address
<substitute email address>. To continue, please enter your password to
link accounts.
```

In the text box, users are expected to enter their IDM account password.

When Multiple Users have the Same Email Address

When you register with a social identity provider, IDM checks the email address of that account against the managed user data store.

If that email address exists for *multiple* IDM managed users, IDM denies the login attempt, with the following error message:

```
Unable to authenticate using login provider
```

IDM denies further attempts to login with that account with the following message:

```
Forbidden request error
```

For information about customizing the End User UI, see the Github repository: ForgeRock/end-user-ui.

The Process for End Users

When your users register with a social identity provider, as defined in "*Social Registration*", they create an account in the IDM managed user data store. As an end user, you can link additional social identity providers to that data store, from the End User UI, using the following steps:

1. Navigate to the End User UI, at an URL such as `http://IDM.example.com:8080`.
2. Log in to the account, either as an IDM user, or with a social identity provider.
3. Navigate to Profile (👤) > Social Sign-in. You should see a list of configured social identity providers.
4. Connect to the social identity providers of your choice. Unless you've already signed in with that social provider, you should be prompted to log in to that provider.
5. To test the result, log out and log back in, using the link for the newly linked social identity provider.

Reviewing Linked Accounts as an Administrator

You can review social identity accounts linked to an IDM account, from the Admin UI and from the command line. You can disable or delete social identity provider information for a specific user from the command line, as described in "Reviewing Linked Accounts Over REST".

When you activate a social identity provider, IDM creates a new managed object for that provider. You can review that managed object in the `managed.json` file, as well as in the Admin UI, by selecting Configure > Managed Objects.

The information shown is reflected in the schema in the `identityProvider-*providername*.json` file for the selected provider.

Note

Do not edit social identity provider profile information through IDM. Any changes that you make won't be synchronized with that provider.

Reviewing Linked Accounts Over REST

To identify linked social identity provider accounts for a user, you must specifically add the `idps` field to your user query. For example, the following query shows bjensen's linked social identity information:

```
curl \
 --header "X-OpenIDM-Username:openidm-admin" \
 --header "X-OpenIDM-Password:openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --request GET \
 "http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+'bjensen'&_fields=idps"
{
  "result": [
    {
      "_id": "bjensen",
      "_rev": "000000003062291c",
      "idps": [
        {
          "_ref": "managed/google/108246554379618660085",
          "_refResourceCollection": "managed/google",
          "_refResourceId": "108246554379618660085",
          "_refProperties": {
            "_id": "ba01a4c3-8a7f-468b-8b09-95f5d34f05ea",
            "_rev": "0000000098619792"
          }
        }
      ]
    }
  ],
  ...
}
```

For more information about a specific social identity provider, query the identity *relationship* using the referred resource ID. The following example shows the information collected from the Google provider for bjensen:

```
curl \
 --header "X-OpenIDM-Username:openidm-admin" \
 --header "X-OpenIDM-Password:openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --request GET \
 "http://localhost:8080/openidm/managed/google/108246554379618660085"
{
  "_id": "108246554379618660085",
  "_rev": "00000000e5cace4d",
  "sub": "108246554379618660085",
  "name": "Barbara Jensen",
  "given_name": "Barbara",
  "family_name": "Jensen",
  "picture": "https://lh3.googleusercontent.com/-XdUIqdMkCWA/AAAAAAAAAAI/AAAAAAAAAAA/4252rscbv5M/
photo.jpg",
  "email": "babs.jensen@gmail.com",
  "email_verified": true,
  "locale": "en",
  "_meta": {
    "subject": "108246554379618660085",
    "scope": [
      "openid",
      "profile",
      "email"
    ],
    "dateCollected": "2018-03-08T02:07:27.882"
  }
}
```

When a user disables logins through one specific social identity provider in the End User UI, that sets `"enabled" : false` in the data for that provider. However, that user's social identity information is preserved.

Alternatively, you can use a REST call to disable logins to a specific social identity provider. The following REST call removes a user's ability to log in through Google:

```
curl  \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --header "Content-type: application/json" \
 --request POST \
 "http://localhost:8080/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?
_action=unbind&provider=google"
```

In this case, the REST call deletes all Google social identity provider information for that user.

Reviewing Linked Accounts From the Admin UI

When you configure a social identity provider, IDM includes two features in the Admin UI.

- The ability to review the social identity accounts linked to specific users. To see how this works, log in to the Admin UI, and select Manage > User, and select a user. Under the Identity Providers tab, you can review the social identity providers associated with a specific account.

- A managed object for each provider. For example, if you've enabled Google as a social identity provider, select Manage > Google. In the screen that appears, you can select the ID for any Google social identity account that has been used or linked to an existing IDM account, and review the profile information shared from that provider.

Manage Social Identity Providers Over REST

You can identify the current status of configured social identity providers with the following REST call:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/authentication
```

The output that you see includes JSON information from each configured social identity provider, as described in the `identityProvider-provider` file in your project's `conf/` subdirectory.

One key line from this output specifies whether the social identity provider is enabled:

```
"enabled" : true
```

If the `SOCIAL_PROVIDERS` authentication module is disabled, you'll see the following output from that REST call:

```
{
   "providers" : [ ]
}
```

For more information, see "Configure the Social Providers Authentication Module".

If the `SOCIAL_PROVIDERS` module is disabled, you can still review the standard configuration of each social provider (enabled or not) by running the same REST call on a different endpoint (do not forget the `s` at the end of `identityProviders`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/identityProviders
```

Note

If you have not configured a social identity provider, you'll see the following output from the REST call on the `openidm/identityProviders` endpoint:

```
{
    "providers" : [ ]
}
```

You can still get information about the available configuration for social identity providers on a slightly different endpoint:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/config/identityProviders
```

The `config` in the endpoint refers to the configuration, starting with the `identityProviders.json` configuration file. Note how it matches the corresponding term in the endpoint.

You can review information for a specific provider by including the name with the endpoint. For example, if you've configured LinkedIn as described in "LinkedIn Social Identity Provider", run the following command:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
http://localhost:8080/openidm/config/identityProvider/linkedIn
```

The above command differs in subtle ways. The `config` in the endpoint points to configuration data. The `identityProvider` at the end of the endpoint is singular, which matches the corresponding configuration file, `identityProvider-linkedIn.json`. And `linkedIn` includes a capital `I` in the middle of the word.

In a similar fashion, you can delete a specific provider:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request DELETE \
http://localhost:8080/openidm/config/identityProvider/linkedIn
```

If you have the information needed to set up a provider, such as the output from the previous two REST calls, you can use the following command to add a provider:

```
curl \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --header "Content-type: application/json" \
 --request PUT \
--data '{
 <Include content from an identityProvider-linkedIn.json file>
}' \
http://localhost:8080/openidm/config/identityProvider/linkedIn
```

IDM incorporates the given information in a file named for the provider, in this case, `identityProvider-linkedIn.json`.

You can even disable a social identity provider with a `PATCH` REST call, as shown:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-type: application/json" \
--request PATCH \
--data '[
   {
      "operation":"replace",
      "field" : "enabled",
      "value" : false
   }
]' \
http://localhost:8080/openidm/config/identityProvider/linkedIn
```

You can reverse the process by substituting `true` for `false` in the previous `PATCH` REST call.

You can manage the social identity providers associated with individual users over REST, as described in "Manage Social Identity Providers Over REST".

Test Social Identity Providers

In all cases, once configuration is complete, you should test the social identity provider. To do so, go through the steps in the following procedure:

1. Navigate to the login screen for the End User UI, `https://openidm.example.com:8443`.
2. Select the `Register` link (after the "Don't have an account?" question) on the login page.
3. You should see a link to sign in with your selected social identity provider. Select that link.

   Note

   If you do not see a link to sign in with any social identity provider, you probably did not enable the option associated with Social Registration. To make sure, access the Admin UI, and select Configure > User Registration.

   Warning

   If you see a redirect URI error from a social identity provider, check the configuration for your web application in the social identity provider developer console. There may be a mistake in the redirect URI or redirect URL.

4. Follow the prompts from your social identity provider to log in to your account.

Note

If there is a problem with the interface to the social identity provider, you might see a Register Your Account screen with information acquired from that provider.

5. Because security questions are enabled by default, you must add at least one security question and answer to proceed. For more information, see "Configure Security Questions".

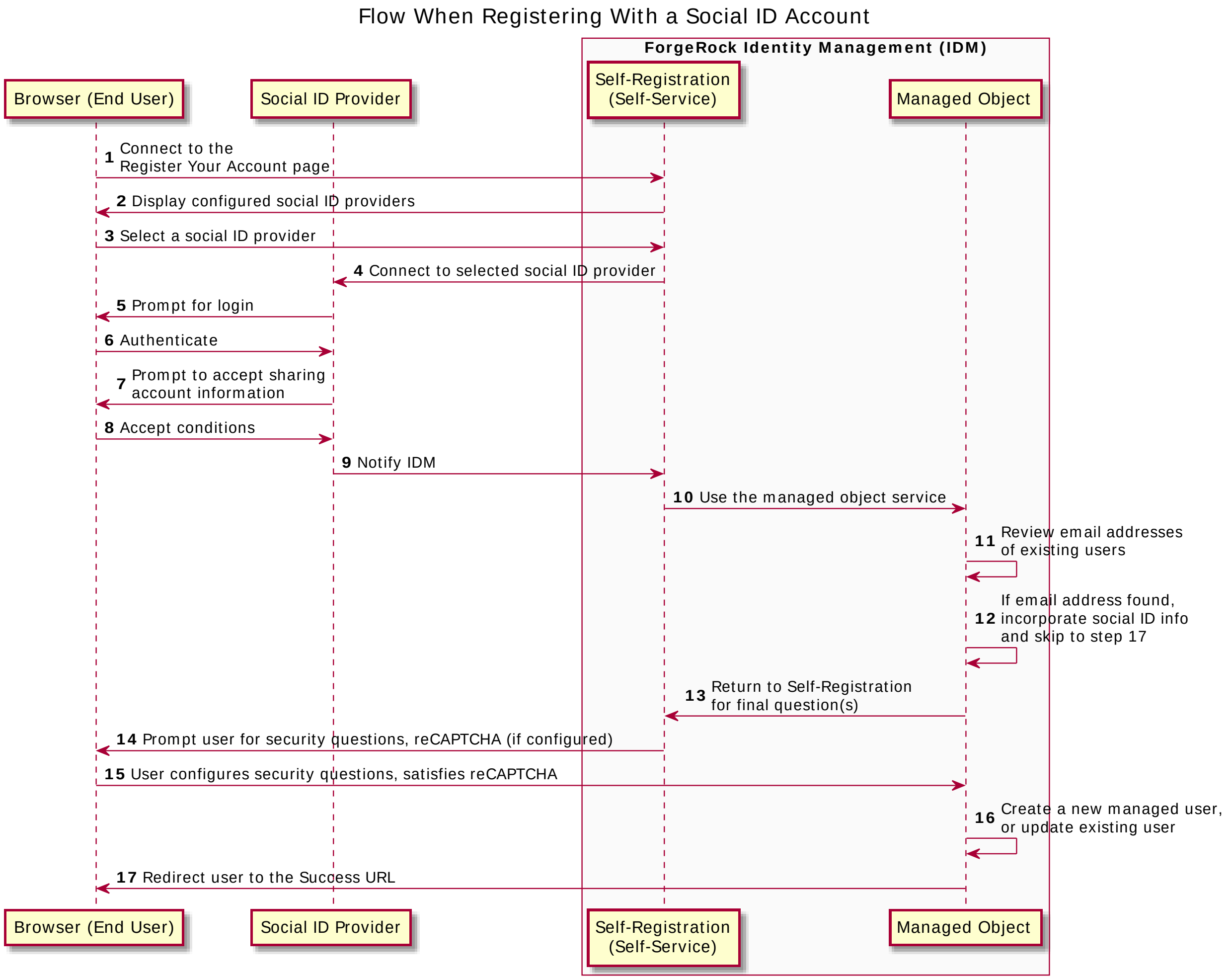
   When the Social ID registration process is complete, you are redirected to the End User UI at `https://openidm.example.com:8443`.

6. You should now be able to use the sign in link for your social identity provider to log in to IDM.

Scenarios When Registering With a Social ID

When users connect to IDM with a social identity provider, it could be the first time they're connecting to your system. They could already have an regular IDM account. They could already have registered with a different social identity provider. This section describes what happens during the self-registration process. The process varies depending on whether there's an existing account in the IDM managed user store.

When Registering Social Identity Providers on IDM


Flow When Registering With a Social ID Account
ForgeRock Identity Management (IDM)
Browser (End User)
Social ID Provider
Self-Registration (Self-Service)
Managed Object
1 Connect to the Register Your Account page
2 Display configured social ID providers
3 Select a social ID provider
4 Connect to selected social ID provider
5 Prompt for login
6 Authenticate
7 Prompt to accept sharing account information
8 Accept conditions
9 Notify IDM
10 Use the managed object service
11 Review email addresses of existing users
12 If email address found, incorporate social ID info and skip to step 17
13 Return to Self-Registration for final question(s)
14 Prompt user for security questions, reCAPTCHA (if configured)
15 User configures security questions, satisfies reCAPTCHA
16 Create a new managed user, or update existing user
17 Redirect user to the Success URL
Browser (End User)
Social ID Provider
Self-Registration (Self-Service)
Managed Object

The following list describes each item in the flow shown in the adjacent figure:

1. From the IDM End User UI, the user selects the `Register` link
2. The self-registration Interface returns a `Register Your Account` page at `http://localhost:8080/#/registration` with a list of configured providers.
3. The user then selects one configured social identity provider.

4. IDM connects to the selected social identity provider.
5. The social identity provider requests end user authentication.
6. The end user authenticates with the social identity provider.
7. The social identity provider prompts the user to accept sharing selected account information.
8. The user accepts the conditions presented by the social identity provider.
9. The social identity provider notifies IDM of the user registration request.
10. IDM passes responsibility to the administrative interface.
11. IDM uses the email address from the social identity provider, and compares it with email addresses of existing managed users.
12. If the email address is found, IDM links the social identity information to that account (and skips to step 16).
13. IDM returns to the self-registration (Self-Service) interface.
14. The self-registration interface prompts the user for additional information, such as security questions, and reCAPTCHA, if configured per "Configure Google reCAPTCHA".
15. The user responds appropriately.
16. IDM creates a new managed user. If the user has already been created, IDM reviews data from the social identity provider, and updates the user data for the managed/*provider* to conform. In this case, the *provider* is a social identity provider such as Google.
17. The user is redirected to the `Success URL`.

Social Identity Widgets

The Admin UI includes widgets that can help you measure the success of your social identity efforts. To add these widgets, take the following steps:

1. Log in to the Admin UI.
2. Select Dashboards, and choose the dashboard to which you want to add the widget.

   For more information about managing dashboards in the UI, see "Manage Dashboards" in the *Setup Guide*.
3. Select Add Widget.

   In the Add Widget window, scroll down to the Social item which includes the following graphical widgets:

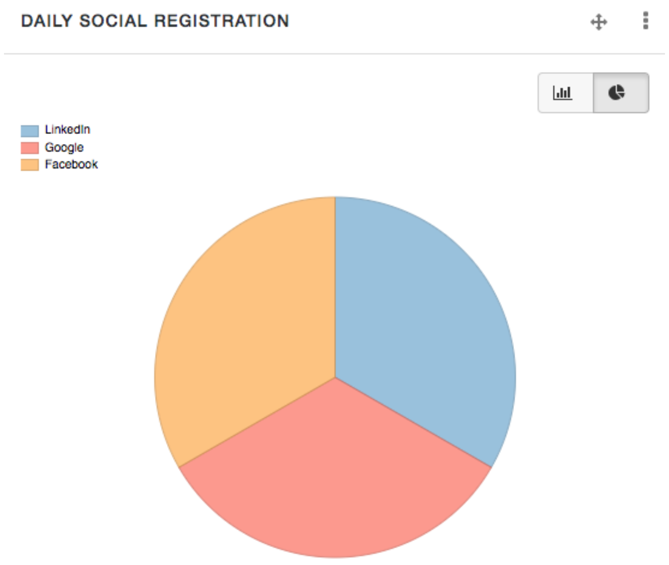
- Social Registration (year)
- Daily Social Registration
- Daily Social Logins

4. Select the widget you want to add and select Settings to configure the widget.

   Optionally, select Preview to see what the widget will look like with the configuration you have applied. Your IDM system must contain some social data to display the preview.

The following example shows daily social registrations, in pie chart form:

Daily Social Registrations on IDM

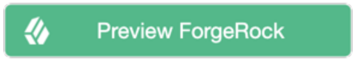
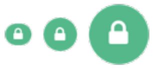


Social Identity Provider Button and Badge Properties

You can configure buttons and badges for each social identity provider, using the Admin UI or by editing the associated `identityProvider-`*`name`*`.json` file. The Admin UI displays examples during social identity provider configuration.

Badges appear in the Admin UI under Configure > Social ID Providers, and in the End User UI under My Account > Sign-in & Security > Social Sign-in.

Buttons appear in the IDM login screens, and when you select Register from the End User UI login screen.

Example Button	Example Badges
	

How IDM displays buttons and badges changes based on how many social identity providers are enabled:

- For up to three social identity providers, IDM displays large buttons, with the text *Register with Provider*.
- For four or more social identity providers, IDM displays smaller buttons with icons.

> **Note**
>
> For seven or more social identity providers, horizontal scrolling may be required.

Properties for Social Identity Provider Buttons and Badges

Property (UI)	Property (JSON file)	Description
Badge background color	`iconBackground`	Color for the social identity provider icon.
Badge icon classname	`iconClass`	Name of the icon class. Can be a Font Awesome name like `fa-google`.
Badge font color	`iconFontColor`	Color for the social identity provider icon font.
Button image path	`buttonImage`	Looks in `openidm/ui/admin/extension` and then `openidm/ui/admin/default` for an image file. Takes precedence over the *Button icon classname*.
Button icon classname	`buttonClass`	Name for the social identity provider class. Can be a Font Awesome name like `fa-yahoo`.
Button display name	`buttonDisplayName`	Name to display on large buttons.
Button styles	`buttonCustomStyle`	Custom styles, such as `background-color: #7B0099; border-color: #7B0099; color:white;`.
Button hover styles	`buttonCustomStyleHover`	Custom styles for the hover state of a button, such as `background-color: #7B0099; border-color: #7B0099; color:white;`.

Chapter 4
Progressive Profile

Progressive profile completion lets you gather profile attributes asynchronously to enrich your users' profile data, and enhance engagement with your customer base. Profile completion requires the creation of one or more *forms* to collect user data.

IDM implements progressive profile completion as a default self-service process. You can use this process as an example of how to build additional functionality into a custom client application, using the Self-Service REST API.

After activating "*Self-Registration*", users need only the following information to register:

- User name
- First name
- Last name
- Email address

Progressive profile completion lets you collect additional information, limited by the attributes defined in the `managed.json` file for your project.

In the following sections, you'll examine how you use progressive profile completion to ask or require more information from users. You're limited only by what properties are defined in your project's `managed.json` file.

Configure a Progressive Profile Completion Form

If you're testing progressive profile completion, you can start from the `selfservice-profile.json` file in the following directory: `openidm/samples/example-configurations/self-service/`

Copy this file to your project's `conf/` directory and start IDM. After the conditions shown in this configuration file are met, end users will see a form prompting them to add a telephone number.

```
{
    "stageConfigs" : [
        {
            "name" : "conditionaluser",
            "identityServiceUrl" : "managed/user",
            "condition" : {
                "type" : "loginCount",
                "interval" : "at",
                "amount" : 25
            },
            "evaluateConditionOnField" : "user",
            "onConditionTrue" : {
                "name" : "attributecollection",
                "identityServiceUrl" : "managed/user",
                "uiConfig" : {
                    "displayName" : "Add your telephone number",
                    "purpose" : "Help us verify your identity",
                    "buttonText" : "Save"
                },
                "attributes" : [
                    {
                        "name" : "telephoneNumber",
                        "isRequired" : true
                    }
                ]
            }
        }
    ]
}
```

The following table includes a detailed list of each property shown in this file:

The `selfservice-profile.json` File

Property	**Description**
`stageConfigs`	Progressive profile completion is a stage of user self-service.
`name`	`conditionaluser` sets up conditions for end users.
`identityServiceUrl`	`managed/user` specifies IDM Managed Users.
`condition`	Condition when to display the form.
`type`	Type of `condition`; for a list of conditions, see "Progressive Profile Completion Conditions".
`evaluateConditionOnField`	IDM evaluates the condition, per `user`.
`onConditionTrue`	Presents the form with the following properties.
`name`	Data that you collect with the form is an `attributeCollection`.
`uiConfig`	Labels to include the in the form seen by the end user.
`displayName`	Form title.
`purpose`	Form explanation.
`buttonText`	Customizable.

Property	Description
`attributes`	Attribute name from `managed.json`.
`isRequired`	If an end user has to enter data to complete a connection to IDM.

The default progressive profile completion process involves two mandatory stages:

- Conditional User Stage
- Attribute Collection Stage

With the previous configuration, users logging in to the End User UI must submit a telephone number on the 25th login.

Progressive Profile Completion Conditions

You can set up a number of different conditions for when users are prompted to add information to their profiles. IDM includes the following pre-defined criteria:

`loginCount`

May specify `at` or `every` number of logins, as defined by the following value: `amount`.

> **Note**
>
> End users can bypass progressive profile completion screens, when configured with a `loginCount`. Every time they see such a request, they can open a new browser window to bypass that request, and log in to the End User UI. They won't have to provide the information requested, even if you've set the attribute as Required under the Attributes tab.

`timeSince`

May specify a time since the user was created, the `createDate`, in `years`, `months`, `weeks`, `days`, `hours`, and `minutes`.

`profileCompleteness`

Based on the number of items completed by the user from `managed.json`, in percent, as defined by `percentLessThan`; for more information, see "Defining Overall Profile Completion".

`propertyValue`

Based on the value of a specific user entry, such as `postalAddress`, which can be defined by "Presence Expressions" in the *Object Modeling Guide*.

Custom Progressive Profile Conditions

You can also set up custom conditions with query filters and scripts. These criteria may deviate from standard query filters described in "Construct Queries" in the *Object Modeling Guide* and standard scripted conditions described in "Add Conditional Policy Definitions" in the *Object Modeling Guide*.

- A queryFilter. For example, the following query filter checks user information for users who live in the city of Portland:

  ```
  "condition" : {
         "type" : "queryFilter",
         "filter" : "/city eq \"Portland\""
         },
  ```

  In addition, you can also reference metadata, as described in "Track User Metadata" in the *Object Modeling Guide*. For example, the following query filter searches for users with:

  - A `loginCount` greater than or equal to five.
  - Does not have a telephone number:

  ```
  "filter" : "(/_meta/loginCount ge 5 and !(/telephoneNumber pr))"
  ```

  Warning

  If you include `_meta` in query filters, the Admin UI will not work for the subject progressive profiling form.

  While it's technically possible to include a number like `5` in the Admin UI with the query filter, IDM would write the number as a string to the `selfservice-profile.json` file. You'd still have to change that number directly in the noted file.

- An inline script (`scripted`), or a reference to a script file; IDM works with scripts written in either JavaScript or Groovy. For example, you could set up a script here:

  ```
  "condition" : {
         "type" : "scripted",
         "script" : {
         "type" : "text/javascript",
         "globals" : { },
         "source" : "<some script code>"
         },
  ```

  Alternatively, you could point to some JavaScript or Groovy file:

  ```
  "condition" : {
         "type" : "scripted",
         "script" : {
         "type" : "text/javascript",
         "globals" : { },
         "file" : "path/to/someScript.js"
         },
  ```

  For the script code, you'll need to reference fields directly, and not by `object.field`. For example, the following code would test for the presence of a telephone number:

  ```
  typeof telephoneNumber === 'undefined' || telephoneNumber === ''
  ```

  While you can also reference metadata for scripts, you can't check for all available fields, as there is no outer `object` field. However, you can refer to fields that are part of the user object.

Configuring Progressive Profile Completion Through the Admin UI

The UI is straightforward; in the Admin UI, when you select Configure > Progressive Profile, you'll add a *New Form*, with:

- Attributes defined in `managed.json`.
- Conditions that may be based on a query filter, a script, or pre-defined criteria such as number of logins.

What you configure in the Admin UI is written to the `selfservice-profile.json` file. The information under the following Admin UI Progressive Profile Completion page tabs is written to the following code blocks in that file:

- Details: `uiConfig`
- Display Condition: `condition`
- Attributes: `attributes`

Warning

When you use the UI, you *must* specify a property under the Attributes tab. Otherwise, IDM won't display a Progressive Profile form. To specify a property, select Configure > Progressive Profile. Select a Progressive Profile form > Attributes tab > Add a Property. Be sure to select an Attribute Name based on user properties configured in the `managed.json` file.

The `auth.profile.json` File

Note

To use `auth.profile.json`, copy the file from `/path/to/openidm/samples/example-configurations/self-service/` to your project's `conf/` directory.

In some circumstances, you may wish to create a temporary role for users who are in the middle of progressive profile completion, such as if you wish to enable access to an endpoint, while prohibiting access to other parts of the End User UI (as well as the rest of IDM).

To do this, you may optionally define an `authenticationRole` in `auth.profile.json`, which you can use as a role assignment in `access.json` or elsewhere.

For example, if you wished to assign access to a custom endpoint for users who have incomplete profiles, you could modify `auth.profile.json` to include a custom `authenticationRole` called `incomplete-profile`:

```
{
    "profileEnhancementProcesses": [
        "selfservice/termsAndConditions",
        "selfservice/kbaUpdate",
        "selfservice/profile"
    ],
    "authenticationRole": "incomplete-profile",
    "authorizationRole": "internal/role/openidm-authorized"
}
```

You could then give access to this role to your custom endpoint in `access.json`:

```
{
    "pattern"    : "endpoint/extra-steps",
    "roles"      : "incomplete-profile",
    "methods"    : "read",
    ...
},
```

Access for these and other roles is governed by the `access.json` script. For more information, see "Configure Access Control in `access.json`" in the *Authentication and Authorization Guide*.

The role specified in `authenticationRole` can be an existing role, or it can be a placeholder string. If it is a placeholder, it will not function as a real role, but can still be used for access in `access.json`, and will appear in access and authentication log files in the `openidm/audit` directory.

Progressive Profile Completion and Metadata

Progressive profile completion requires that you track object metadata. Configure tracking of the following data:

- `createDate`: The date the user was created; used in the `onCreateUser.js` script in the `openidm/bin/defaults/script` directory.
- `loginCount`: The number of logins, by user.
- `stagesCompleted`: Used to track progressive profile forms, and whether they've been completed, by user.

User acceptance of Terms & Conditions is tracked by default (see "Terms & Conditions").

Defining Overall Profile Completion

A user profile is based on every item in `managed.json` where both `viewable` and `userEditable` are set to `true`. Every qualifying item has equal weight.

So, if there are 20 qualifying items in `managed.json`, a user who has entries for 10 items has a *Profile completion percentage* of 50.

REST Requests in a Progressive Profile Completion Process

The following REST requests and responses demonstrate the flow through a profile completion process, given the previous configuration:

1. Client attempts a login for the 25th time:

```
curl \
 --header "X-OpenIDM-Username: bjensen" \
 --header "X-OpenIDM-Password: Passw0rd" \
 --header "X-OpenIDM-NoSession: false" \
 --request POST \
 "https://localhost:8443/openidm/authentication?_action=login"
{
  "_id": "login",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "processesRequired": true,
    "component": "managed/user",
    "authLogin": true,
    "authenticationIdProperty": "username",
    "roles": [],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "protectedAttributeList": ["password"],
    "requiredProfileProcesses": ["selfservice/profile"],
    "id": "51c6c46d-3d7b-4671-8295-0c8ee39e8549",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  },
  "authenticationId": "bjensen"
}
```

Note

The values of the `requiredProfileProcesses` and `roles` properties in the returned output trigger the remainder of the process. If `requiredProfileProcesses` is present and not empty, there are processes that must be completed. Ultimately, the process must return a full access `role` (such as `internal/role/openidm-authorized`) and continue to the user profile page.

2. Server sends a GET request to the `profile` endpoint and returns `"type": "conditionaluser"` and `"tag": "initial"` to start the profile completion process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request GET \
 "https://localhost:8443/openidm/selfservice/profile"
{
 "_id": "1",
 "_rev": "991096945",
 "type": "conditionaluser",
 "tag": "initial",
 "requirements": {
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "Attribute Details",
  "type": "object",
  "properties": {},
  "attributes": [{
   "name": "telephoneNumber",
   "isRequired": true,
   "schema": {
    "type": "string",
    "title": "Telephone Number",
    "description": "Telephone Number",
    "viewable": true,
    "userEditable": true,
    "pattern": "^\\+?([0-9\\- \\(\\)])*$",
    "usageDescription": "",
    "isPersonal": true
   },
   "value": null
  }],
  "uiConfig": {
   "displayName": "Add your telephone number",
   "purpose": "Help us verify your identity",
   "buttonText": "Save"
  }
 }
}
```

3. Client submits requirements, in this case, the required profile field. Server response includes `"tag": "end"` and `"success": true` to signal the end of the profile process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request POST \
 --data '{
     "input":{
         "attributes":{
             "telephoneNumber":"555-555-1234"
         }
     }
 }'
 "https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
 "type": "conditionaluser",
 "tag": "end",
 "status": {
  "success": true
 },
 "additions": {}
}
```

Viewing Profile Completeness

You can view how complete a profile is, presented as the percentage of user-editable attributes that have been filled out on a profile. To do so, send a REST call to the `selfservice/profile/completeness` endpoint:

```
curl \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --request GET \
"http://localhost:8080/openidm/selfservice/profile/completeness/managed/user/3a8cabef-
d4a3-4f60-926a-52f27257bde6"
{
  "_id": "managed/user/3a8cabef-d4a3-4f60-926a-52f27257bde6",
  "_rev": "00000000c38d9344",
  "completeness": 42.857143
}
```

Chapter 5
Password Reset

IDM supports self-service user password reset. When enabled, users who forget their passwords can log in to the IDM End User UI, and can verify their identities with options such as email validation and security questions.

You can also generate random passwords when users are created. For more information, see "Generating Random Passwords" in the *Security Guide*.

Password reset lets registered users reset their own passwords. The following stages can be included in a password reset process:

- Captcha Stage (optional)
- User Query Stage (mandatory)
- Email Validation Stage (optional)
- KBA Security Answer Verification Stage (optional)
- Password Reset Stage (mandatory)

If all of these stages are configured, the password reset configuration (in `conf/selfservice-profile.json` looks similar to the following:

+ *Example password reset configuration*

```
{
    "stageConfigs" : [
        {
            "name" : "captcha",
            "recaptchaSiteKey" : "...",
            "recaptchaSecretKey" : "...",
            "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
        },
        {
            "name" : "userQuery",
            "validQueryFields" : [
                "userName",
                "mail",
                "givenName",
                "sn"
            ],
            "identityIdField" : "_id",
            "identityEmailField" : "mail",
            "identityUsernameField" : "userName",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "emailValidation",
            "identityEmailField" : "mail",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@example.com",
            "subject" : "Reset password email",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Reset your password",
                "fr" : "Réinitialisez votre mot de passe"
            },
            "messageTranslations" : {
                "en" : "...Click to reset your password...",
                "fr" : "...Cliquez pour réinitialiser votre mot de passe..."
            },
            "verificationLinkToken" : "%link%",
            "verificationLink" : "https://localhost:8443/#/passwordreset/"
        },
        {
            "name" : "kbaSecurityAnswerVerificationStage",
            "kbaPropertyName" : "kbaInfo",
            "identityServiceUrl" : "managed/user",
            "kbaConfig" : null
        },
        {
            "name" : "resetStage",
            "identityServiceUrl" : "managed/user",
            "identityPasswordField" : "password"
        }
    ],
    "snapshotToken" : {
        "type" : "jwt",
        "jweAlgorithm" : "RSAES_PKCS1_V1_5",
        "encryptionMethod" : "A128CBC_HS256",
        "jwsAlgorithm" : "HS256",
        "tokenExpiry" : "300"
    },
    "storage" : "stateless"
}
```

User Password Reset Configuration Files

To set up basic user password reset features, you'll need at least the following configuration files:

- `selfservice-reset.json`

  You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`.

- `ui-configuration.json`

  You can find this file in the default IDM project configuration directory, `openidm/conf`.

To set up self-service user password reset registration, enable the following boolean in `ui-configuration.json`:

```
"passwordReset" : true,
```

You can include several features with user password reset, as shown in the following excerpts of the `selfservice-reset.json` file:

- If you've activated Google reCAPTCHA for user self-service registration, you'll see the following code block:

  ```
  {
      "name" : "captcha",
      "recaptchaSiteKey" : "<siteKey>",
      "recaptchaSecretKey" : "<secretKey>",
      "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
  },
  ```

  As suggested by the code, you'd substitute the actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, see "Configure Google reCAPTCHA".

- For password reset, IDM needs to verify user identities. To ensure that password reset links are sent to the right user, include the following code block:

  ```
  {
      "name" : "userQuery",
      "validQueryFields" : [
          "userName",
          "mail",
          "givenName",
          "sn"
      ],
      "identityIdField" : "_id",
      "identityEmailField" : "mail",
      "identityUsernameField" : "userName",
      "identityServiceUrl" : "managed/user"
  },
  ```

  This code lets IDM verify user identities by their username, email address, first name (`givenName`), or last name (`sn`, short for surname).

- If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json`, see "Configuring Emails for Password Reset".

- If you've configured security questions, users who self-register will have to create questions and answers during the self-registration process.

  If the feature is enabled, users who've been reconciled from external data stores will also be prompted, once, upon their next login, to add security questions and answers. The relevant code block is shown here, which points IDM to other configuration files as discussed in links from this section.

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Configuring Password Reset From the Admin UI

To configure Password Reset from the Admin UI, select Configure > Password Reset. When you select Enable Password Reset, you'll see a `Configure Password Reset Form` that lets you specify the:

- Identity Resource, typically `managed/user`
- Advanced Options, Snapshot Token, typically a JSON Web Token (JWT)
- Advanced Options, Token Lifetime, with a default of 300 seconds

You can also add these settings to the following configuration file: `selfservice-reset.json`. When you modify these settings in the Admin UI, IDM creates the file for you.

Configuring Emails for Password Reset

To configure emails for password reset, you can add the following code block to the `selfservice-reset.json` file:

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "subject" : "Reset password email",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Reset your password",
        "fr" : "Réinitialisez votre mot de passe"
    },
    "messageTranslations" : {
        "en" : "<h3>Click to reset your password</h3><h4><a href=\"%link%\">Password reset link</a></h4>",
        "fr" : "<h3>Cliquez pour réinitialiser votre mot de passe</h3><h4><a href=\"%link%\">Mot de passe
 lien de réinitialisation</a></h4>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/passwordreset/"
},
```

As suggested by the code block, it includes default password reset email messages in English (`en`) and French (`fr`). The `verificationLink` sent with the email takes users to the IDM password reset URL.

As noted in "REST Requests in a Password Reset Process", you can make these changes over the following endpoint URI: `/openidm/config/selfservice/reset`

If desired, you can also configure self-service password reset emails through the Admin UI. Select Configure > Password Reset. If needed, activate the Enable Password Reset option, and in the Email Validation box, select the ✎ icon. The Configure Validation Email pop-up dialog box should appear.

When you use the Admin UI to customize password reset emails, you can review the changes in the `selfservice-reset.json` file.

REST Requests in a Password Reset Process

The following REST requests and responses demonstrate the flow through a simple password reset process. To keep the process simple, this flow does not include the Google ReCAPTCHA stage, or the Security Answer Verification stage:

1. Client initiates the password reset,

   The server returns the `initial` tag:

```
curl \
--request GET \
"https://localhost:8443/openidm/selfservice/reset"
{
  "type": "parameters",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Parameters",
    "type": "object",
    "properties": {
      "returnParams": {
        "description": "Parameter named 'returnParams'",
        "type": "string"
      }
    }
  }
}
```

2. Initial requirements submission with an empty payload.

   The server returns requirements for the `userQuery` stage, and the JWT:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "input":{}
}' \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "userQuery",
  "tag": "initial",
  "requirements": {
    "$schema": "http:\/\/json-schema.org\/draft-04\/schema#",
    "description": "Find your account",
    "type": "object",
    "required": [
      "queryFilter"
    ],
    "properties": {
      "queryFilter": {
        "description": "filter string to find account",
        "type": "string"
      }
    }
  },
  "token": "eyJ0e...FYkE"
}
```

3. The client provides the requirements for the `userQuery` stage, along with the JWT. The process progresses to the `emailValidation` stage:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "token": "eyJ0e...FYkE",
  "input": {"queryFilter": "userName eq \"bjensen\""}
}' \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "emailValidation",
  "tag": "validateCode",
  "requirements": {
    "$schema": "http:\/\/json-schema.org\/draft-04\/schema#",
    "description": "Verify emailed code",
    "type": "object",
    "required": [
      "code"
    ],
    "properties": {
      "code": {
        "description": "Enter code emailed",
        "type": "string"
      }
    }
  },
  "token": "eyJ0e...FYkE"
}
```

The server converts that requirement and token to a URL that is emailed.

4. The user receives an email with the password reset link.

   Clicking the link sends another POST request to the `emailValidation` stage, along with the token, and a `code`:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/#/passwordreset/&token=eyJ0e...FYkE&code=code"
```

   The process advances to the reset stage and returns its requirements.

5. After email validation, the client submits the new password. The process advances to the reset stage, updates the managed object, and exits:

```
curl \
--header "X-OpenIDM-Username: anonymous" \
--header "X-OpenIDM-Password: anonymous" \
--request POST \
--header "Content-Type: application/json" \
--data {
  "token": "eyJ0e...FYkE",
  "input": {
    "password": "Passw0rd"
  }
} \
"https://localhost:8443/openidm/selfservice/reset?_action=submitRequirements"
{
  "type": "resetStage",
  "tag": "end",
  "status": {
    "success": true
  },
  "additions": {
  }
}
```

Chapter 6 Username Retrieval

Username retrieval lets registered users retrieve a forgotten username, based on the provision of alternative information in the user record, such as email address, last name, or given name. Depending on how this process is configured, the retrieved username can be emailed to the user or displayed directly.

The REST requests in this section assume that the username is emailed to the user, and that the configuration is similar to that in the example configuration file (`samples/example-configurations/self-service/selfservice-username.json`):

+ *Example username retrieval configuration*

```
{
    "stageConfigs" : [
        {
            "name" : "userQuery",
            "validQueryFields" : [
                "mail",
                "givenName",
                "sn"
            ],
            "identityIdField" : "_id",
            "identityEmailField" : "mail",
            "identityUsernameField" : "userName",
            "identityServiceUrl" : "managed/user"
        },
        {
            "name" : "emailUsername",
            "emailServiceUrl" : "external/email",
            "emailServiceParameters" : {
                "waitForCompletion" : false
            },
            "from" : "info@admin.org",
            "mimeType" : "text/html",
            "subjectTranslations" : {
                "en" : "Account Information - username"
            },
            "messageTranslations" : {
                "en" : "<h3>Username is:</h3><br />%username%"
            },
            "usernameToken" : "%username%"
        },
        {
            "name" : "retrieveUsername"
        }
    ],
    "storage" : "stateless"
}
```

Username Retrieval Configuration

To set up basic forgotten username configuration, you'll need at least the following configuration files:

- `selfservice-username.json`

  You can find a template version of this file in the following directory: `openidm/samples/example-configurations/self-service`.

- `ui-configuration.json`

  You can find this file in the default IDM project configuration directory, `openidm/conf`.

To set up forgotten username retrieval, enable the following boolean in `ui-configuration.json`:

```
"forgotUsername" : true,
```

You can include several features with forgotten username retrieval, as shown in the following excerpts of the `selfservice-reset.json` file:

- If you've activated Google reCAPTCHA for forgotten username retrieval, you'll see the following code block:

  ```
  {
      "name" : "captcha",
      "recaptchaSiteKey" : "<siteKey>",
      "recaptchaSecretKey" : "<secretKey>",
      "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
  },
  ```

  As suggested by the code, you'd substitute actual `siteKey` and `secretKey` assigned by Google for your domain. For more information, see "Configure Google reCAPTCHA".

- For forgotten username retrieval, IDM needs to verify user identities. To ensure that usernames are sent to the right user, include the following code block:

  ```
  {
      "name" : "userQuery",
      "validQueryFields" : [
          "mail",
          "givenName",
          "sn"
      ],
      "identityIdField" : "_id",
      "identityEmailField" : "mail",
      "identityUsernameField" : "userName",
      "identityServiceUrl" : "managed/user"
  },
  ```

This code allows IDM to verify user identities by their username, email address, first name (`givenName`), or last name (`sn`, short for surname).

- If you have included email verification, you must configure an outgoing email server. For details about the required addition to `selfservice-registration.json`, see "Configuring Emails for Forgotten Username".
- The following code block, after confirming user identity, allows IDM to display the username:

```
{
    "name" : "retrieveUsername"
}
```

Configuring Forgotten Username Retrieval From the Admin UI

To configure forgotten username retrieval from the Admin UI, select Configure > Forgotten Username. When you select Enable Forgotten Username Retrieval, you'll see a *Configure Forgotten Username Form* that allows you to specify the:

- Identity Resource, typically `managed/user`
- Advanced Options, Snapshot Token, typically a JSON Web Token (JWT).
- Advanced Options, Token Lifetime, with a default of 300 seconds

You can also add these settings to the following configuration file: `selfservice-username.json`. When you modify these settings in the Admin UI, IDM creates the file for you.

Configuring Emails for Forgotten Username

To configure emails for forgotten username functionality, you can add the following code block to the `selfservice-username.json` file:

```
{
    "name" : "emailUsername",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@example.com",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Account Information - username"
    },
    "messageTranslations" : {
        "en" : "<h3>Username is:</h3><br />%username%"
    },
    "usernameToken" : "%username%"
},
```

As suggested by the code block, it includes default email messages in English (`en`), with a `usernameToken` that includes the actual username in the message.

As noted in "*Username Retrieval*", you can make these changes over the following endpoint URI: `/openidm/config/selfservice/username`

If desired, you can also configure forgotten username retrieval emails through the Admin UI. Select Configure > Forgotten Username. If needed, activate the Enable Forgotten Username Retrieval option, and in the Email Username box, select the ✎ icon. The Configure Email Username pop-up should appear.

When you use the Admin UI to customize forgotten username retrieval emails, you can review the changes in the `selfservice-username.json` file.

REST Requests in a Forgotten Username Process

The following REST requests and responses demonstrate the flow through a forgotten username process:

1. Client initiates the username retrieval process. The server returns the initial set of requirements:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --header "X-OpenIDM-NoSession: true" \
 --request GET \
 "https://localhost:8443/openidm/selfservice/username"
{
   "_id":"1",
   "_rev":"959264722",
   "type":"userQuery",
   "tag":"initial",
   "requirements":{
      "$schema":"http://json-schema.org/draft-04/schema#",
      "description":"Find your account",
      "type":"object",
      "required":[
         "queryFilter"
      ],
      "properties":{
         "queryFilter":{
            "description":"filter string to find account",
            "type":"string"
         }
      }
   }
}
```

2. Client submits the requirements, along with the token. Server returns the username and the `end` tag to indicate the end of the process:

```
curl \
 --header "X-OpenIDM-Username: anonymous" \
 --header "X-OpenIDM-Password: anonymous" \
 --request POST \
 --data '{
  "token": "eyJ0eXAiOiJKV1QiLCJjdHkiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ZXlKMGVY...W5ywOcr8",
  {
     "input":{
        "queryFilter":"mail eq \"babs.k.jensen@gmail.com\""
  }
}' \
 "https://localhost:8443/openidm/selfservice/username?_action=submitRequirements"
{
   "type":"retrieveUsername",
   "tag":"end",
   "status":{
      "success":true
   },
   "additions":{
      "userName":"bjensen"
   }
}
```

Chapter 7

Additional Configuration

This chapter describes additional configuration options for user self-service.

Additional Configuration Options

Email Notification	Privacy & Consent	UMA & Trusted Devices
Configure Notification Emails.	Configure Privacy and Consent.	Set Up User-Managed Access (UMA), Trusted Devices, and Privacy.
Terms & Conditions	User Self-Service Tokens	End User UI Notifications
Configure Terms & Conditions.	Tokens and User Self-Service.	Configure End User UI Notifications.
reCAPTCHA	Identity Fields	Security Questions
Configure Google reCAPTCHA.	Configure Identity Field Associations.	Configure Security Questions (KBA).
Custom Policies	End User UI	
Add Custom Policies for Self-Registration and Password Reset.	Configure Self-Service End User UI.	

Configure Notification Emails

When you configure the outbound email service, IDM can use that service to notify users of significant events, primarily related to user self-service. For specifics, see the following table for related notification emails:

Configuring Notification Emails

Situation	Configuration File	Details
When a user is successfully registered	`emailTemplate-welcome.json`	See "User Self-Registration Email Template"
When a user asks for their forgotten username	`selfservice-username.json`	See "Configuring Emails for Forgotten Username"
When a user registers using self-service and needs to verify their email address	`selfservice-registration.json`	See "Configuring Emails for Self-Service Registration"
When a user asks for a password reset	`selfservice-reset.json`	See "Configuring Emails for Password Reset"

Each email template can specify an email address to use in the `From` field. If this field is left blank, IDM will default to the address specified in Email Settings.

Note

Email templates utilize Handlebar expressions to reference object data dynamically. For example, to reference the `userName` of an object:

```
{{object.userName}}
```

Note

Some email providers, such as Google, will override the `From` address you specify in the templates, and instead use the address used to authenticate with the SMTP server. The email address specified in the template may still be present, but in an email header hidden from most users, such as `X-Google-Original-From`.

User Self-Registration Email Template

When a new user registers through the IDM self-registration interface (and if you have configured outbound email), that user will get a welcome email as configured in the `emailTemplate-welcome.json` file:

```
{
    "enabled" : true,
    "from" : "",
    "subject" : {
        "en" : "Your account has been created"
    },
    "message" : {
        "en" : "<html><body><p>Welcome to OpenIDM. Your username is '{{object.userName}}'.</p></body></
html>"
    },
    "defaultLocale" : "en"
}
```

You may want to make the following changes:

- Add an email address to the `from` property, perhaps an email address for your organization's systems administrator.
- Set up appropriate locale(s).
- Modify the subject line as needed.
- Include a welcome `message` appropriate to your organization.

Managing Email Templates from the Admin UI

The Admin UI includes tools that can help you customize email messages related to two administrative tasks: creating users and resetting passwords.

To configure these messages from the Admin UI, select Configure > Email Settings > Templates, where you'll see the following option:

- Welcome: To configure emails that notify a user of a newly created account, as defined in `emailTemplate-welcome.json`.

  Note

  IDM sends the same welcome email to users created with a REST call. For an example of user creation over REST, see "Managed Users" in the *Object Modeling Guide*.

Configure Privacy and Consent

As an end user, you might want to control what happens to your personal data. For IDM, that means control of how your data is shared with external systems. The example in "Marketo Connector" in the *Connectors Guide* shows how you can generate a marketing leads database, only for those users who have selected a specific preference. Also read "Configure Privacy and Consent".

IDM allows you to regulate access to two different kinds of personal data:

- *User information*: while marketers want user information such as addresses and telephone numbers, IDM allows you to let individual users decide whether to share that data. For more information, see "Regulating HTTP Access to Personal Data".
- *Account information*: by default, IDM prevents REST-based access to passwords with the `private` scope, as defined in the `managed.json` file. You can extend this protection to other properties. For more information, see "Restricting HTTP Access to Sensitive Data".

You can configure Privacy and Consent for users who register directly through IDM, or through a social identity provider. For more information on the registration process, see "Configure User Self-Registration" and "*Social Registration*".

When you have configured Privacy and Consent, end users must agree to share their data before they can obtain a registered account.

To configure Privacy and Consent, edit the following configuration files:

- In `selfservice-registration.json`, add the following JSON object:

  ```
  {
      "name" : "consent",
      "consentTranslations" : {
       "en" : "<Substitute appropriate Privacy and Consent wording>",
       "fr" : "<Substitute appropriate Privacy and Consent wording, in French>"
      }
  },
  ```

  Add custom privacy and consent notices for all your required languages in the `consentTranslations` property.

  Alternatively, send the corresponding request over REST to the `/openidm/config/selfservice/registration` endpoint.

- In the mapping configuration, include:

  ```
  "consentRequired" : true,
  ```

+ *To Configure Privacy and Consent in the Admin UI*

1. Select Configure > Mappings, and select the mapping for which you want to configure Privacy and Consent.

   Important

   Although the Admin UI includes the Privacy & Consent switch for all mappings, it makes sense to configure Privacy and Consent *only* for mappings *from* the Managed Object source *to* an external target resource. In other words, end users give their consent to transfer some or all of their managed user data to an external system.

2. On the Advanced tab of the mapping, select Enable Privacy & Consent, then select Save.

3. Select Configure > User Registration, and Enable User Registration (if it is not already enabled).

4. On the Options tab, select Privacy & Consent, then add custom privacy notices for all required languages.

Regulating HTTP Access to Personal Data

In some cases, you might want to allow users to choose whether to share their personal data. "Configure User Preferences" describes how to allow users to select basic preferences for updates and marketing. They can select these preferences when they register and in the End User UI.

Examine the `managed.json` file for your project. Every relevant property should include two settings that determine whether a user can choose to share or not share that property:

- `isPersonal`: When set to `true`, specifies personally identifying information. By default, the `isPersonal` option for `userName` and `postalAddress` is set to `true`.

  `usageDescription`: Includes additional information that can help users understand the sensitivity of a specific property such as `telephoneNumber`.

The `consentedMappings` property in a managed user object enables the user to specify an array of mappings (target systems) with which they consent to sharing their identifying information. The following sample excerpt of the default managed user object schema shows the `consentedMappings` property definition:

```
"consentedMappings": {
    "title": "Consented Mappings",
    "description": "Consented Mappings",
    "type": "array",
    "viewable": false,
    "searchable": false,
    "userEditable": true,
    "usageDescription": "",
    "isPersonal": false,
    "items": {
      "type": "object",
      "title": "Consented Mapping",
      "properties": {
        "mapping": {
          "title": "Mapping",
          "description": "Mapping",
          "type": "string",
          "viewable": true,
          "searchable": true,
          "userEditable": true
        },
        "consentDate": {
          "title": "Consent Date",
          "description": "Consent Date",
          "type": "string",
          "viewable": true,
          "searchable": true,
          "userEditable": true
        }
      },
      "order": [
        "mapping",
        "consentDate"
      ],
      "required": [
        "mapping",
        "consentDate"
      ]
    },
    "returnByDefault": false,
    "isVirtual": false
}
```

Restricting HTTP Access to Sensitive Data

You can protect specific sensitive managed data by marking the corresponding properties as `private`. Private data, whether it is encrypted or not, is not accessible over the REST interface. Properties that are marked as private are removed from an object when that object is retrieved over REST.

To mark a property as private, set its `scope` to `private` in the `conf/managed.json` file.

The following extract of the `managed.json` file shows how HTTP access is prevented on the `password` property:

```
{
    "objects": [
        {
            "name": "user",
            "schema": {
                "id" : "http://jsonschema.net",
                "title" : "User",
                ...
                "properties" : {
                    ...
                    "password" : {
                        "title" : "Password",
                        ...
                        "encryption" : {
                            "purpose": "idm.password.encryption"
                        },
                        "scope" : "private",
                        ...
        }
    ]
}
```

Tip

To configure private properties by using the Admin UI:

1. Select Configure > Managed Objects, and select the object type whose property values you want to make private (for example User).
2. On the Properties tab, select the property that must be private and select the Private checkbox.

A potential caveat relates to private properties. If you use an HTTP `GET` request, you won't even see private properties. Even if you know all relevant private properties, a `PUT` request would replace the entire object in the repository. In addition, that require would effectively remove all private properties from the object. To work around this limitation, use a `POST` request to update only those properties that require change.

For example, to update the `givenName` of user jdoe, you could run the following command:

```
$ curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '[
  {
    "operation": "replace",
    "field": "/givenName",
    "value": "Jon"
  }
]' \
"https://localhost:8443/openidm/managed/user?_action=patch&_queryId=for-userName&uid=jdoe"
```

Note

The filtering of private data applies only to direct HTTP read and query calls on managed objects. No automatic filtering is done for internal callers, and the data that these callers choose to expose.

Configure UMA, Trusted Devices, and Privacy

In the following sections, you will refer to AM documentation to set up User-Managed Access (UMA), Trusted Devices, and Privacy for your end users. The section requires IDM authentication with AM bearer tokens and the `rsFilter` authentication module. For more information, see "Authenticate through AM" in the *Authentication and Authorization Guide*.

Tip

If you want to configure both UMA and Trusted Devices in AM, configure these features in the following order, as described in the sections that follow:

1. Set up UMA
2. Use AM to configure UMA-based resources
3. Configure Trusted Devices

If you have to reconfigure UMA at a later date, you'll have to first disable Trusted Devices. You can enable Trusted Devices, once again, afterwards.

User Managed Access in IDM

When you integrate IDM with ForgeRock Access Management (AM) you can take advantage of AM's abilities to work with User-Managed Access (UMA) workflows. AM and IDM use a common installation of ForgeRock Directory Services (DS) to store user data.

When you have configured IDM to authenticate through AM bearer tokens, you can configure AM to work with UMA. For more information, see the AM User-Managed Access (UMA) Guide. From that guide, you need to know how to:

- Set up AM as an authorization server.
- Register resource sets and client agents in AM.
- Help users manage access to their protected resources through AM.

Pay close attention to the AM documentation on configuring an OAuth 2.0 UMA Client and UMA Server. You may need to add specific grant types to each OAuth 2.0 application.

If you follow AM documentation to set up UMA, you'll see instructions on setting up users as resource owners and requesting parties. If you set up users in AM, be sure to include the following information for each user:

- First Name
- Last Name
- Email Address

AM writes this information to the common DS user data store. You can then synchronize these users to the IDM Managed User data store, with a command such as:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
"http://localhost:8080/openidm/recon?_action=recon&mapping=systemLdapAccounts_managedUser"
```

After your users have shared UMA resources from the AM Self-Service UI, they can view what they've done and shared in the IDM End User UI, by selecting the Sharing icon (➦).

Configuring Trusted Devices on IDM

You can configure Trusted Devices through AM, using the following sections of the AM Authentication and Single Sign-On Guide: *Configuring Authentication Chains* and *Device ID (Match) Authentication Module*. You can use the techniques described in these sections to set up different authentication chains for administrators and regular users.

You can create an AM authentication chain with the following modules and criteria:

AM Authentication Chain Modules

Module	Criteria
Data Store	Requisite
Device Id (Match)	Sufficient
Device Id (Save)	Required

This is different from the authentication chain described in the following section of the *AM Authentication and Single Sign-On Guide*: Device ID (Match) Authentication Module, as it does not include the HOTP Authentication Module.

When trusted devices are enabled, users are presented with a prompt on a screen with the following question "Add to Trusted Devices?". If the user selects `Yes`, that user is prompted for the name of the Trusted Device.

Note

In default configurations, trusted devices are not saved for the AM `amadmin` account. However, you can set up different AM administrative users as described in the following section of the AM *Setup and Maintenance Guide*: *Delegating Realm Administration Privileges*.

You can set up different authentication chains for regular and administrative users, as described in the AM *Authentication and Single Sign-On Guide*.

Terms & Conditions

Most entities require users to accept Terms & Conditions. By default, this feature is active for user self-registration in IDM. When a user accepts Terms & Conditions, IDM records relevant information in the `_meta` data for that user, as described in "Identifying When a User Accepts Terms & Conditions".

Note

To use this feature, auth.profile.json must be present in the `/path/to/openidm/conf/` directory.

Terms & Conditions Configuration Files

`selfservice.terms.json`

Exists in the `/path/to/openidm/conf/` directory and contains the default Terms & Conditions language:

```
{
  "versions": [
    {
      "version": "0.0",
      "termsTranslations": {
        "en": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
 incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
 ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
 voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
 proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
      },
      "createDate": "2019-10-28T04:20:11.320Z"
    }
  ],
  "active": "0.0",
  "uiConfig": {
    "displayName": "We've updated our terms",
    "purpose": "You must accept the updated terms in order to proceed.",
    "buttonText": "Accept"
  }
}
```

`selfservice-termsAndConditions.json`

To force existing IDM users to accept new Terms & Conditions during login, copy `selfservice-termsAndConditions.json` from your project's `conf` directory to your project directory, and edit the file, as necessary.

The following example applies Terms & Conditions to the `managed/user` store:

```
{
  "stageConfigs" : [
    {
      "name" : "conditionaluser",
      "identityServiceUrl" : "managed/user",
      "condition" : {
        "type" : "terms"
      },
      "evaluateConditionOnField" : "user",
      "onConditionTrue" : {
        "name" : "termsAndConditions"
      }
    },
    {
      "name" : "patchObject",
      "identityServiceUrl" : "managed/user"
    }
  ]
}
```

Note

IDM does not support `<form>` elements or `<script>` tags in Terms & Conditions text.

Substitute Terms & Conditions content to meet the legal requirements of your applicable governing entities.

`selfservice.terms.json` Details

Property	Description
`version`	Specifies a version number (must be unique).
`termsTranslations`	Supports Terms & Conditions in different languages. **Note** For Terms & Conditions in multiple languages, what the end user sees depends on their browser default language, based on ISO-639 language codes: First, IDM determines the active version, as defined in the `selfservice.terms.json` file: • If the browser default language matches one of the configured Terms & Conditions languages, IDM displays it. • If the browser default language does not match any configured Terms & Conditions languages: ◦ IDM displays the `en` language. ◦ If there is no `en` language, IDM displays the first configured language for the active version.
`createDate`	Creation date.
`active`	Specifies the version of Terms & Conditions shown to users; must match an existing `version`.
`displayName`	The title of the Terms & Conditions page, as seen by end users.
`purpose`	Help text shown below the `displayName`.
`buttonText`	Button text shown to the end user for acceptance.

Preview Terms & Conditions as an End User

To preview Terms & Conditions in the End User UI:

1. Create a regular user.
2. Log in to the End User UI as the new user.

   IDM prompts you to accept the default Terms & Conditions.

Updating Terms & Conditions over REST

You can manage the configuration for Terms & Conditions over the following endpoints:

- `openidm/config/selfservice.terms`
- `openidm/config/selfservice/termsAndConditions`

For example, the following command would replace the value of `buttonText`:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
   "operation" : "replace",
   "field" : "uiConfig/buttonText",
   "value" : "OK"
} ]' \
"http://localhost:8080/openidm/config/selfservice.terms"
```

Identifying When a User Accepts Terms & Conditions

You can identify when a user accepts Terms & Conditions, as well as the associated version. To do so, take the following steps:

- If needed, find identifying information for all managed users:

  ```
  curl \
  --header "X-OpenIDM-Username: openidm-admin" \
  --header "X-OpenIDM-Password: openidm-admin" \
  --header "Accept-API-Version: resource=1.0" \
  --request GET \
  "http://localhost:8080/openidm/managed/user?_queryId=query-all"
  ```

- Use REST to get a specific user's information. This example illustrates how a user with a `userName` of `kvaughan` has already accepted a specific version of Terms & Conditions:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=userName+eq+'kvaughan'&_fields=*,/_meta/*"
{
  "result": [
    {
      ...
      "userName": "kvaughan",
      ...
        "termsAccepted": {
          "acceptDate": "2018-04-12T22:55:33.370Z",
          "termsVersion": "2.0"
        },
        "createDate": "2018-04-12T22:55:33.395Z",
        "lastChanged": {
          "date": "2018-04-12T22:55:33.395Z"
        },
        "loginCount": 1,
        "_rev": "00000000776f8be1",
        "_id": "69124007-05ec-46e1-a8a8-ecc3d94db124"
      }
    }
  ],
  ...
}
```

Configuring Terms & Conditions in the Admin UI

From the Admin UI, select Configure > Terms & Conditions. You can then create a new version, which prompts you to configure the following:

- Version number (must be unique).
- If there are existing Terms & Conditions, you'll see a *Make active* switch for the new Terms & Conditions.
- Locale, in ISO-639 format.
- Terms & Conditions, in the specified language locales. You can set up Terms & Conditions in text and/or basic HTML.

Once you've added Terms & Conditions in all desired locales, select Save to save them in the `selfservice.terms.json` file.

Note

The Admin UI does not let you delete existing Terms & Conditions.

Once you have at least one set of Terms & Conditions, you should see a Settings tab, where you can:

- Require acceptance; the next time any end user logs into IDM, that user will see a copy of your Terms & Conditions, with the Header, Description, and Button Text.
- To make sure new users have to accept these Terms & Conditions, select Configure > User Registration in the Admin UI. Enable Terms & Conditions under the Options tab. For more information, see "*Self-Registration*". Users who self-register will see the following message, with a link to those Terms & Conditions:

```
By creating an account, you agree to the Terms & Conditions
```

These changes are recorded in `_meta` data for each user, and can be retrieved through REST calls described in "Identifying When a User Accepts Terms & Conditions".

Tokens and User Self-Service

Many processes within user self-service involve multiple stages, such as user self-registration, password reset, and forgotten username. As the user transitions from one stage to another, IDM uses JWT tokens to represent the current state of the process. As each stage is completed, IDM returns a new token. Each request that follows includes that latest token.

For example, users who use these features to recover their usernames and passwords get two tokens in the following scenario:

- The user goes through the forgotten username process, gets a JWT Token with a lifetime (default = 300 seconds) that lets the user get to the next step in the process.
- With username in hand, that user may then start the password reset process. That user gets a second JWT token, with the token lifetime configured for that process.

Note

The default IDM JWT token is encrypted and stateless. However, if you need a token that can be included in a link that works in all email clients, change the `snapshotToken` `type` in the appropriate configuration file to `uuid`.

End User UI Notifications

Whenever there are changes related to individual users, IDM sends notifications to the affected user. When the user logs in to the End User UI, they can find their notifications by selecting the bell (🔔) icon.

Notifications are configured in `notification-*.json` files, as described in "Custom Notifications" in the *Audit Guide*.

IDM includes a `notifications` endpoint that can help you identify all notifications:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/internal/notification?_queryFilter=true"
```

To list notifications by user ID, include the `_notifications` field in a query on that ID:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/managed/user/e3a9385b-733f-4a1c-891b-c89292b30d70?_fields=_notifications/*"
```

You can filter notifications with any of the properties shown in the following table:

End User Notification Properties

Property	Description
`createDate`	Creation date
`notificationType`	Message type: limited to info, warning, or error
`message`	Message seen by the end user

You can get additional information from the activity audit log, in the `audit/activity.audit.json` file, including the following:

- The `userId` who made the change.
- The `runAs` name of the user who made the change.
- If configured in "Fields to Watch", any watched fields that have changed.
- If the password was changed, as indicated by the `passwordChanged` property.

Configure Google reCAPTCHA

Google reCAPTCHA helps prevent bots from registering users or resetting passwords on your system. For Google documentation on this feature, see *Google reCAPTCHA*. IDM works with Google reCAPTCHA v2.

To use Google reCAPTCHA, you will need a Google account and your domain name (RFC 2606-compliant URLs such as `localhost` and `example.com` are acceptable for test purposes). Google then provides a site key and a secret key that you can include in the self-service function configuration.

For example, you can set up reCAPTCHA by adding the following code block to the configuration file for user self-registration `selfservice-registration.json`, password reset, `selfservice-reset.json`, and forgotten username `selfservice-username.json` functionality.

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "< Insert Site Key Here >",
    "recaptchaSecretKey" : "< Insert Secret Key Here >",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

You may also add the reCAPTCHA keys through the UI for each of these self-service features.

Configure Identity Fields

It is possible to adjust the property associated with a field in user self-service. Properties that are used by self-service functions can be set using identity field properties in your configuration. For example, if you had changed the `mail` property in `managed/user` to instead be `email`, you would then update `identityEmailField` in your self-service configuration to be `"identityEmailField" : "email",`. There are currently six identity fields that can be customized:

- `identityServiceUrl` - sets where self-service stores and retrieves its data, such as `managed/user`.
- `identityUsernameField` - sets the property associated with the username of the user.
- `identityEmailField` - sets the property associated with the email address of the user.
- `identityPasswordField` - sets the property associated with the password of the user.
- `identityIdField` - sets the property associated with the ID of the user, which is used when performing user queries.
- `identityAccountStatus` - sets the property associated with the account status of the user, which is used when performing user queries.

Not every identity field is used in each self-service stage. For more information about which fields are required for each stage, see "*Self-Service Stage Reference*".

Note

If you have removed usernames from your `managed/user` schema in favor of using another property (such as email), you will still need to set `identityUsernameField` to the new property in order for self-service to function correctly.

Configure Security Questions

IDM uses security questions to let users verify their identities. Security questions are sometimes referred to as Knowledge-Based Authentication (KBA). When an administrator has configured security questions, self-service users can choose from the questions set in the `selfservice.kba.json` file, as described in "Security Questions and Self-Registration".

You can prompt users to update their security questions. As these questions may be subject to risks, you can set up IDM to prompt the user to update and/or add security questions, courtesy of the `selfservice-kbaUpdate.json` file. For more information, see "Prompt to Update Security Questions".

Security Questions and Self-Registration

The user is prompted to enter answers to pre-configured or custom security questions, during the self-registration process. These questions are used to help verify an identity when a user requests a password reset. These questions do not apply for users who need username retrieval.

The template version of the `selfservice.kba.json` file includes `minimumAnswersToDefine`, which requires a user to define at least that many security questions and answers, along with `minimumAnswersToVerify`, which requires a user to answer (in this case), at least one of those questions when asking for a password reset.

```
{
    "kbaPropertyName" : "kbaInfo",
    "minimumAnswersToDefine": 2,
    "minimumAnswersToVerify": 1,
    "questions" : {
        "1" : {
            "en" : "What's your favorite color?",
            "en_GB" : "What is your favourite colour?",
            "fr" : "Quelle est votre couleur préférée?"
        },
        "2" : {
            "en" : "Who was your first employer?"
        }
    }
}
```

You can change or add questions in JSON format, or if you're configuring user self-registration, you can also edit these questions through the Admin UI. From the Admin UI, select Configure > User Registration. Enable User Registration, select Options > Security Questions, and select the edit icon to add, edit, or delete these questions.

Any change you make to the security questions under User Registration also applies to Password Reset. To confirm, select Configure > Password Reset. Enable Password Reset, and edit the Security Questions. You'll see the same questions there.

In addition, individual users can configure their own questions and answers:

- During the user self-registration process.
- From the End User UI, in the user's Profile section (👤), under Account Security > Security Questions.

Important

A managed user's security questions can only be changed through the `selfservice/userupdate` endpoint, or when the user is created through `selfservice/registration`, and provides their own questions. You cannot manipulate a user's `kbaInfo` property directly through the `managed/user` endpoint.

When the answers to security questions are hashed, they are converted to lowercase. If you intend to pre-populate answers with a mapping, the `openidm.hash` function, or the `secureHash` mechanism, you must provide the string in lowercase to match the value of the answer.

KBA Answer Hashing

By default, KBA answers are SHA-256 hashed upon save. To specify another type of hashing, edit the self-service KBA configuration *(You can edit the self-service KBA configuration over REST at the config/selfservice.kba endpoint, or directly in the conf/selfservice.kba.json file.)*:

+ *Using the Filesystem*

Add the secureHash property to the `conf/selfservice.kba.json` file:

```
"secureHash" : {
  "algorithm": "{type}",
  "configProp": value
}
```

For example, to use BCRYPT hashing:

```
"secureHash": {
  "algorithm": "BCRYPT",
  "cost": 13
}
```

+ *Using REST*

1. Get the current self-service KBA configuration *(You can edit the self-service KBA configuration over REST at the config/selfservice.kba endpoint, or directly in the conf/selfservice.kba.json file.)*:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/config/selfservice.kba"
{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What's your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  }
}
```

2. Add the `secureHash` property for the alternative hashing, and replace the self-service KBA configuration *(You can edit the self-service KBA configuration over REST at the config/selfservice.kba endpoint, or directly in the conf/selfservice.kba.json file.)*. For example, to use BCRYPT hashing:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What'\''s your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  },
  "secureHash": {
    "algorithm": "BCRYPT",
    "cost": 13
  }
}' \
"http://localhost:8080/openidm/config/selfservice.kba"
{
  "_id": "selfservice.kba",
  "kbaPropertyName": "kbaInfo",
  "minimumAnswersToDefine": 2,
  "minimumAnswersToVerify": 1,
  "questions": {
    "1": {
      "en": "What's your favorite color?",
      "en_GB": "What is your favourite colour?",
      "fr": "Quelle est votre couleur préférée?"
    },
    "2": {
      "en": "Who was your first employer?"
    }
  },
  "secureHash": {
    "algorithm": "BCRYPT",
    "cost": 13
  }
}
```

Supported Hashing Algorithms and Configuration Properties

Algorithm	Config Property and Description
`BCRYPT`	`cost` - Value between 4 and 31. Default is `13`.
`PBKDF2`	`hashLength` - Byte-length of the generated hash. Default is `16`. `iterations` - Number of cryptographic iterations. Default is `20000`. `hmac` - HMAC algorithm. Default is `SHA3-256`.
`SCRYPT`	`hashLength` - Byte-length of the generated hash, must be greater than or equal to 8. Default is `16`. `n` - CPU/Memory cost parameter. Must be greater than 1, a power of 2, and less than *2^(128 * r / 8)*. Default is `32768`. `p` - Parallelization parameter. Must be a positive integer less than or equal to *Integer.MAX_VALUE / (128 * r * 8)*. Default is `1`. `r` - Block size. Must be greater than or equal to 1. Default is `8`.
`SHA-256`	This is the default hashing.
`SHA-384`	N/A
`SHA-512`	N/A

KBA Attempts Account Lockout

To configure account lockout based on the security questions, add the following lines to your `selfservice.kba.json` file:

```
"numberOfAttemptsAllowed" : 2,
"kbaAttemptsPropertyName" : "lockoutproperty"
```

With this configuration, users who make more than two mistakes in answering security questions are prevented from using the password reset facility until the `kbaAttemptsPropertyName` field is removed, or the number is set to a value lower than the `numberOfAttemptsAllowed`. The number of mistakes is recorded in whatever property you assign to `kbaAttemptsPropertyName` (`lockoutproperty`, in this example).

If you are using an explicit mapping for managed user objects, you must add this *lockoutproperty* to your database schema *and* to the `objectToColumn` mapping in your repository configuration file.

For example, the previous configuration would require the following addition to your `conf/repo.jdbc.json` file:

```
"explicitMapping" : {
    "managed/user": {
        "table" : "managed_user",
        "objectToColumn": {
            ...
            "lockoutproperty" : "lockoutproperty",
            ...
        }
```

You would also need to create a `lockoutproperty` column in the `openidm.managed_user` table, with datatype `VARCHAR`. For example:

```
mysql> show columns from managed_user;

+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| objectid             | varchar(38) | NO   | PRI | NULL    |       |
| rev                  | varchar(38) | NO   |     | NULL    |       |
| username             | varchar(255) | YES  | UNI | NULL    |       |
| password             | varchar(511) | YES  |     | NULL    |       |
| accountstatus        | varchar(255) | YES  | MUL | NULL    |       |
| postalcode           | varchar(255) | YES  |     | NULL    |       |
| lockoutproperty      | varchar(255) | YES  |     | NULL    |       |
...
```

Warning

Once you deploy these IDM self-service features, you should never remove or change existing security questions, as users may have included those questions during the user self-registration process.

Prompt to Update Security Questions

IDM supports a requirement for users to update their security questions, in the `selfservice-kbaUpdate.json` file. You can find this file in the following directory: `/path/to/openidm/samples/example-configurations/self-service`.

Alternatively, if you set up security questions from the Admin UI, you can navigate to Configure > Security Questions > Update Form, and select Enable Update. This action adds a `selfservice-kbaUpdate.json` file to your project's `conf/` subdirectory.

For more information on this configuration file, see "Conditional User Stage".

Add Custom Policies for Self-Registration and Password Reset

IDM defines policies for usernames and passwords, in the `openidm/bin/defaults/script/policy.js` file. To enforce these policies for user self-registration and password reset, add the following objects to your `conf/policy.json` file, under `resources`:

```
{
    "resource" : "selfservice/registration",
    "calculatedProperties" : {
        "type" : "text/javascript",
        "source" : "require('selfServicePolicies').getRegistrationProperties()"
    }
},
{
    "resource" : "selfservice/reset",
    "calculatedProperties" : {
        "type" : "text/javascript",
        "source" : "require('selfServicePolicies').getResetProperties()"
    }
},
```

Self-Service End User UI

This chapter includes the steps that you would take to verify functionality from an end user point of view. Some of the options described in this chapter can be used to help support compliance with the General Data Protection Regulation (GDPR).

For information about customizing the End User UI, see the Github repository: ForgeRock/end-user-ui.

Localizing the End User UI

The End User UI is configured in US English. For more information on how to localize and modify the messages in the End User UI, see the following section of the ForgeRock Identity Management (End User) repository on " Translations and Text".

Change the End User UI Path

By default, the End User UI is registered at the root context and is accessible at the URL `https://localhost:8443`. To specify a different URL, edit the `*project-dir*/conf/ui.context-enduser.json` file, setting the `urlContextRoot` property to the new URL.

For example, to change the End User UI URL to `https://localhost:8443/exampleui`, edit the file as follows:

```
"urlContextRoot" : "/exampleui",
```

Alternatively, to change the End User UI URL in the Admin UI, follow these steps:

1. Log in to the Admin UI.
2. Select Configure > System Preferences, and select the Self-Service UI tab.
3. Specify the new context route in the Relative URL field.

Provide a Logout URL to External Applications

By default, an End User UI session is invalidated when a user clicks on the Log out link. In certain situations, external applications might require a distinct logout URL to which users can be routed, to terminate their UI session.

The logout URL is `#logout`, appended to the UI URL; for example, `https://localhost:8443/#logout/`.

The logout URL effectively performs the same action as clicking on the Log out link of the UI.

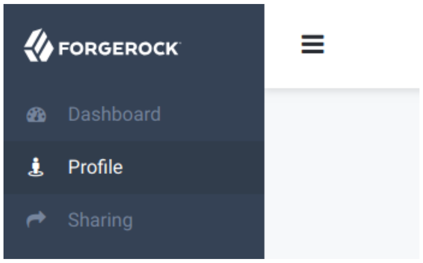
Privacy: My Account Information in the End User UI

While end users can find their information in the End User UI, you can use REST calls and audit logs to find the same information. Some of the information in this section, such as Trusted Devices and UMA-based sharing, might require integration with ForgeRock Access Management (AM), as described in the Platform Setup Guide.

What the enduser sees upon log in to the End User UI depends on which features are configured.

- When you log in to the End User UI, you'll be taken to the IDM Profile page (👤), with at least the following information under settings:
  - Account Security
  - Preferences
  - Account Controls
- You'll see at least a Dashboard (⏲) and a Profile icon (👤) in the left hand pane. If you've configured UMA as described in "Configure UMA, Trusted Devices, and Privacy", you'll also see a Sharing icon (↪). To see descriptions with each icon, select the Menu icon (☰):

Icons in the End User UI



- When you add features described earlier in this chapter, you'll see additional options in the profile page, as described in the following table:

Information in the End User Profile Page

Title	Description	Section
Account Security	Password and Security Questions, default	"Configure Security Questions"
Social Sign-in	Links to Social Identity Provider Accounts	"*Social Registration*"
Authorized Applications	Applications that can access an account	"Authorized Applications"
Trusted Devices	Based on system and browser	"Configuring Trusted Devices on IDM"
Preferences	Default	"Configure User Preferences"
Personal Data Sharing	Provides control	"Personal Data Sharing"
Account Controls	Includes collected account data (Default)	"Account Controls"

Personal Information

End users can find their account details in the End User UI, by selecting the Profile icon (👤) > Edit Personal Info. By default, user information includes at least the following properties: Username, First Name, Last Name, and Email Address.

Each user can modify this information as needed, as long as `"userEditable" : true` for the property in your project's `managed.json` file. For more information, see "Create and Modify Object Types" in the *Object Modeling Guide*.

Sign-In & Security

Under this tab, end users can change their passwords. They can also add, delete, or modify security questions, and link or unlink supported social identity accounts. For more information, see "Configure Security Questions" and "*Social Registration*".

Preferences

The preferences tab allows end users to modify marketing preferences, as defined in the `managed.json` file, and the Managed Object User property Preferences tab. For more information, see "Configure User Preferences".

End users can toggle marketing preferences. When IDM includes a mapping to a marketing database, these preferences are sent to that database. This can help administrators use IDM to target marketing campaigns and identify potential leads.

Trusted Devices

A *trusted device* uses AM's Device ID (Match) and Device ID (Save) authentication modules, as described in the AM Authentication and Single Sign-On Guide. When such modules are configured (see "Configuring Trusted Devices on IDM"), end users can add such devices the first time they log in from a new location.

During the login process, when an end user selects *Log In*, that user is prompted for a *Trusted Device Name*. Users see their added devices under the Trusted Devices tab.

A trusted device entry is paired with a specific browser on a specific system. The next time the same end user logs in from the same browser and system, in the same location, that user should not be prompted to enter a trusted device again.

End users can remove their trusted devices from the tab.

Authorized Applications

The Authorized Applications section is specific to end users as OAuth 2 clients. and reflects the corresponding section of the AM Self-Service dashboard, as described in the following section of the AM *OAuth 2.0 Guide on*: *User Consent Management*.

Personal Data Sharing

This section assumes that as an administrator, you've followed the instructions in "Configure Privacy and Consent" to enable Privacy & Consent.

End users who see a Personal Data Sharing section have control of whether personal data is shared with an external database, such as one that might contain marketing leads.

The managed object record for end users who consent to sharing such data is shown in REST output and the audit activity log as one `consentedMappings` object:

```
"consentedMappings" : [ {
   "mapping" : "managedUser_systemLdapAccounts",
   "consentDate" : "2017-08-25T18:13:08.358Z"
}
```

If enabled, end users will see a Personal Data Sharing section in their profiles. If they select the Allow link, they can see the data properties that would be shared with the external database.

This option supports the right to restrict processing of user personal data.

Account Controls

The Account Controls section allows end users to download their account data (in JSON format), and to delete their accounts from IDM.

Important

When end users delete their accounts, the change is propagated to external systems by implicit sync. However, it is then up to the administrator of the external system to make sure that any additional user information is purged from that system.

To modify the message associated with the `Delete Your Account` option, refer to the section about Translations in the README of the public ForgeRock Identity Management (End User) Git repository. Find the `translation.json` file, search for the `deleteAccount` code block, and edit the information.

The options shown in this section can help meet requirements related to data portability, as well as the right to be forgotten.

Chapter 8

Custom Self-Service Stages

This chapter demonstrates how to build, deploy, and configure a custom stage, and how to add it to a self-service process. You can use the classes in the sample project as a basis to develop your own stages.

To implement a custom stage in the End User UI, see the following instructions from the ForgeRock End User UI Git Repository: *How to Add a Self-Service Stage to the UI*.

Sample Stage

ForgeRock provides a sample custom stage project with the minimum classes and project file required for any self-service stage. The sample project has a dependency on the `forgerock-selfservice-core` artifact. Engage ForgeRock support for access to the required repositories.

The sample project implements a stage named `MathProblem`, which generates a simple math problem that must be completed in order to progress to the next stage.

The project includes the following files, required for any custom self-service stage:

A Maven project file (`pom.xml`)

Pay particular attention to the `maven-bundle-plugin` in this file:

```
<plugins>
    <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
            <instructions>
                <Fragment-Host>org.forgerock.openidm.selfservice</Fragment-Host>
            </instructions>
        </configuration>
    </plugin>
</plugins>
```

This plugin indicates that Apache Felix should attach the custom stage artifact to IDM’s self-service bundle.

A *configuration class*

(`src/main/java/org/forgerock/selfservice/custom/MathProblemStageConfig.java`)

The configuration class reads configuration data from a corresponding configuration (JSON) file. The class represents each configuration item for the stage as properties of the class.

An *implementation class*

(`src/main/java/org/forgerock/selfservice/custom/MathProblemStage.java`)

The implementation class is the main orchestration class for the stage.

Build the Sample Stage

To build the sample stage, you must have Apache Maven installed.

1. Clone the ForgeRock Selfservice Custom Stage repository.
2. Change to the root directory of the project you cloned:

```
cd /path/to/forgerock-selfservice-custom-stage
```

3. This version of IDM works with version `26.1.x` of ForgeRock Commons. Locate the latest version `26.1.x` tag:

   a. List the latest tags for this version:

   ```
   git tag --list | grep 26.1
   26.1.0-20210407090058-81dd8fe
   ...
   26.1.0-latest
   ```

   b. Check out the latest version:

   ```
   git checkout -b test tags/26.1.0-latest
   Switched to a new branch 'test'
   ```

4. Build the sample stage:

```
mvn clean install
```

This build process creates the `forgerock-selfservice-custom-stage/self-service/forgerock-selfservice -custom-stage/target/forgerock-selfservice-custom-stage-version.jar` file.

5. Copy the compiled stage to the `openidm/bundle` directory:

```
cp target/forgerock-selfservice-custom-stage-version.jar /path/to/openidm/bundle/
```

6. Restart IDM.

Creating a Configuration for the Sample Stage

To create a configuration for this stage, examine the configuration class (`MathProblemStageConfig.java`). Three configuration properties must be specified in the corresponding configuration file:

- `class`

  For the default IDM self-service stages, you specify the stage `name` in the configuration, in the format `"name" : "stage-name"`. For example:

  ```
  "name" : "captcha"
  ```

  For custom stages, you must specify the stage configuration class, in the format `"class" : "stage_config_classname"`. For example:

  ```
  "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig"
  ```

- `leftValue`
- `rightValue`

The configuration for this stage will therefore look something like the following:

```
{
    "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig",
    "leftValue" : int,
    "rightValue" : int
},
```

Important

When you write a custom stage, the `equals` and `hashCode` methods must be overridden to include local class members.

Testing the Custom Stage

Stages are implemented as part of a *self-service process*. For more information, see "The Self-Service Process Flow". To test your custom stage, you need to add it to a self-service process. You can create a new process, or use one of the default processes available through the Admin UI.

In this example, we add the custom stage to the User Registration process and test it as part of self-registration, as follows:

1. In the Admin UI Select Configure > User Registration > Enable to enable user registration.

   This step creates a `selfservice-registration.json` file in your project's `conf` directory. There are a number of stages in that process by default, for example, the `parameters` stage:

```
"stageConfigs" : [
    {
        "name" : "parameters",
        "parameterNames" : [
            "returnParams"
        ]
    },

...
]
```

2. Add your custom stage to the process by creating a configuration item in the `stageConfigs` array:

   ```
   "stageConfigs" : [
       {
           "name" : "parameters",
           "parameterNames" : [
               "returnParams"
           ]
       },
       {
       "class" : "org.forgerock.selfservice.custom.MathProblemStageConfig",
       "leftValue" : 12,
       "rightValue" : 4
       },
   ...
   ]
   ```

   Note that self-service stages can generally not be configured in random order. For example, some stages require input from the process `state` that has been populated by a preceding stage. For the purposes of this example, add the `MathProblem` stage directly after the `parameters` stage.

3. Disable *all-in-one* registration.

   By default, the registration phase has *all-in-one* registration enabled. All-in-one registration covers a number of registration stages. For the purposes of testing the custom stage, disable all-in-one registration by setting `"allInOneRegistration" : false` in `selfservice-registration.json`. For more information, see "All-In-One Registration".

4. Save the changes to the `selfservice-registration.json` file.

   IDM reloads the configuration automatically—you do not need to restart the server.

5. Log in to the End User UI (at `https://localhost:8443/` by default) and select Register.

   The first stage to be displayed should be the Math Problem you configured previously.

Appendix A. Self-Service Stage Reference

This chapter describes the individual stages that can be called by a self-service process, the purpose of the stage, any required parameters, dependencies on preceding or following stages, and the expected stage output.

The stages are listed in alphabetical order, for ease of reference, but they cannot be configured in random order. For example, some stages require input from the process `state` that has been populated by a preceding stage.

The `identityServiceURL` is a required parameter for most self-service stages. The self-service stages operate on a managed object. The `identityServiceURL` indicates the object type, for example, `managed/user`.

All-In-One Registration

A registration process that consists of more than one stage can include an optional "*super stage*" named `allInOneRegistration`, that is set outside of the `stageConfigs` array as follows:

```
"allInOneRegistration" : true
```

All-in-one registration covers a number of registration stages. If this property is `true`, in the registration process configuration, IDM scans the configuration for any of the following stages:

- `parameters`
- `captcha`
- `termsAndConditions`
- `kbaSecurityAnswerDefinitionStage`

- `consent`
- `idmUserDetails`

If any of these stages are found, the individual stages are effectively removed from the configuration, and a new configuration is generated that accumulates all the found stages.

The purpose of all-in-one registration is to obtain a set of initial requirements, then to advance to the end of all six stages simultaneously. This lets self-registration be completed on a single registration form. As the process advances, it gathers any output, errors, and others from all six stages (or however many stages have been configured). The process then returns whatever was gathered from the cumulative stages, including any outstanding requirements. Depending on the output, the process might be required to go through the stages more than once, as the outstanding requirements are provided.

Important

All-in-one registration requires multiple registration stages. If your registration process includes only one stage, for example, `consent`, `allInOneRegistration` must be set to `false`, to preserve the registration flow.

If all-in-one registration is `false`, any additional stages listed in the registration process (`selfservice-registration.json`) must be listed *after* the parameters and `idmUserDetails` stages. If a stage occurs before the `idmUserDetails` stage without all-in-one registration, both social and regular registration will not work.

OpenAM Auto-Login Stage

This stage is used to perform auto-login when IDM is configured with ForgeRock Access Management (AM). The stage is similar to the local auto-login stage, but also requires the `returnParams` stored in `state` (populated in the Parameters Stage).

Example configuration

```
{
    "name" : "openAmAutoLogin",
    "identityUsernameField": "userName",
    "identityPasswordField": "password",
    "openAMBaseUrl" : "http://AM.example.com:8080/openam/",
    "authenticationEndpoint" : "json/realms/root/authenticate"
}
```

Dependencies

This stage should appear towards the end of a process—it cannot be the first stage in a process.

Required Parameters

- `authenticationEndpoint` - the AM Authentication Endpoint URL.
- `openAMBaseUrl` - the URL of the AM server.

- `identityUsernameField` - the managed object property that contains the username.
- `identityPasswordField` - the managed object property that contains the user password.

Attribute Collection Stage

The purpose of this stage is to collect managed object properties to insert into the user profile. The list of properties to be collected is defined as part of the configuration.

This stage updates the managed object directly, and checks whether attributes are *required*. If required attributes are not provided, the stage returns the list of requirements again. This stage can throw an exception if there is an error attempting to save the updated attributes.

Example configuration

```
{
    "name" : "attributecollection",
    "identityServiceUrl" : "managed/user",
    "uiConfig" : {
        "displayName" : "Add your telephone number",
        "purpose" : "Help us verify your identity",
        "buttonText" : "Save"
    },
    "attributes" : [
        {
            "name" : "telephoneNumber",
            "isRequired" : true
        }
    ]
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts
- `uiConfig` - how the requirements list is conveyed to an end user
- `attributes` - the array of attributes to be collected. For each attribute, the `isRequired` parameter indicates whether the attribute is mandatory for the stage to proceed.

Captcha Stage

This stage verifies a `response` variable populated in `state` by the reCaptcha mechanism. If the response is missing, or if validation fails (typically, if the configuration does not include the required reCaptcha configuration parameters), the stage throws a bad request exception. If validation succeeds, the process advances to the next stage.

Example configuration

```
{
    "name" : "captcha",
    "recaptchaSiteKey" : "6LdahVIUAAAAAJcwGTWdl40sG9tpdgFIyZKUSzyU",
    "recaptchaSecretKey" : "6LdahVIUAAAAANF-017E-b8PyBqLrhLaOHUX8ch-",
    "recaptchaUri" : "https://www.google.com/recaptcha/api/siteverify"
},
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- `recaptchaSiteKey` - invokes the reCAPTCHA service
- `recaptchaSecretKey` - authorizes communication between IDM and the reCAPTCHA server to verify the user's response
- `recaptchaUri` - the reCaptcha verification API

Conditional User Stage

Defines a condition, that results in a boolean (`true` or `false`). The outcome of the condition determines which stage should be executed next.

Example configuration

```
{
    "name": "conditionaluser",
    "identityServiceUrl": "managed/user",
    "condition": {
        "type": "kbaQuestions"
    },
    "evaluateConditionOnField": "user",
    "onConditionFalse": {
        "name": "kbaUpdateStage",
        "kbaConfig": null,
        "identityServiceUrl" : "managed/user",
        "uiConfig" : {
            "displayName" : "Update your security questions",
            "purpose" : "Please review and update your security questions",
            "buttonText" : "Update"
        }
    }
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process. If the condition evaluates to `true`, the process moves on to the next stage.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts.
- `condition` - the condition type, which can be one of the following:
  - `kbaQuestions` - a boolean (`true` or `false`) that indicates whether configured security questions have been answered.
  - `queryFilter` - a common filter expression such as `"filter" : "/co eq \"US\""`.
  - `script` - lets you configure a custom scripted condition.
  - `loginCount` - a condition based on the number of password or social authentication-based login requests.
  - `terms` - a boolean (`true` or `false`) that indicates whether configured Terms and Conditions have been accepted.
  - `timesincelogin` - sets a condition based on the period of time since the last login, in years, months, weeks, days, hours, and minutes.
- `evaluateConditionOnField` - the property on which the condition should be evaluated.
- `onConditionFalse` - the details of the stage to be called if the condition evaluates to false.

Consent Stage

This stage evaluates a boolean `consentGiven` (`true` or `false`). The user is prompted to consent for each mapping that is set to require consent. If consent is required but not given, the stage fails with an exception. It is up to the client to handle that exception, for example, to prevent registration if the user does not provide consent.

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process.

Required Parameters

- None.

Email Validation Stage

This stage retrieves the email address from `state` (or in response to initial requirements), then verifies the validity of the email address with the user who submitted the requirements through an email process.

Example configuration

```
{
    "name" : "emailValidation",
    "identityEmailField" : "mail",
    "emailServiceUrl" : "external/email",
    "emailServiceParameters" : {
        "waitForCompletion" : false
    },
    "from" : "info@admin.org",
    "subject" : "Reset password email",
    "mimeType" : "text/html",
    "subjectTranslations" : {
        "en" : "Reset your password",
        "fr" : "Réinitialisez votre mot de passe"
    },
    "messageTranslations" : {
        "en" : "Click to reset your password <a href=\"%link%\">Password reset link</a>",
        "fr" : "Cliquez pour réinitialiser votre mot de passe<a href=\"%link%\">Mot de passe lien de
 réinitialisation</a>"
    },
    "verificationLinkToken" : "%link%",
    "verificationLink" : "https://localhost:8443/#/passwordreset/"
},
```

Dependencies

This stage expects a preceding stage to populate the user email address in `state`. The stage has no downstream dependencies.

Required Parameters

- Email configuration. For more information, see "Configuring Emails for Self-Service Registration".

IDM User Details Stage

This stage collects new user data and stores it in `state`. This is the only stage that sets up a user from nothing. The stage does not *create* a managed object directly—it simply gathers and stores the data. The Self-Registration Stage consumes the stored user data and creates the managed object from it.

The IDM User Details stage executes multiple times, requesting additional requirements each time. There are different ways for the stage to advance, depending on how the user create request is initiated.

If the user completes a self-service registration form, the input contains a `user` object, collected from the form, and populates that user in `state`. If the user registers through social authentication, the stage reads the profile from the remote identity provider, normalizes it, then maps it to a user object. That user object is then put into `state`.

If the new user object in `state` is incomplete or does not meet policy requirements, the stage returns a new set of requirements, indicating the collected data and the missing data. The registering user is requested to submit the additional data, then the stage revalidates the object in `state`. When all of the required data to register a user is present, the process advances to the next stage.

Important

The user data remains in `state`—no managed user object is created.

Example configuration

```
{
    "name" : "idmUserDetails",
    "identityEmailField" : "mail",
    "socialRegistrationEnabled" : true,
    "identityServiceUrl" : "managed/user",
    "registrationProperties" : [
        "userName",
        "givenName",
        "sn",
        "mail"
    ],
    "registrationPreferences": ["marketing", "updates"]
},
```

Dependencies

This stage *must* occur in any registration process. It has no dependencies on previous stages but must have the Self-Registration Stage somewhere downstream in the process, to create the managed user object.

Required Parameters

- `identityEmailField` - the attribute on the managed user object that contains the user email.
- `identityServiceUrl` - the managed object type on which this stage acts.
- `socialRegistrationEnabled` - optional, `false` if not specified. Indicates whether the stage must read the user profile from a remote identity provider and normalize it.
- `registrationProperties` - an array of properties that must be provided by a registering user in order for the stage to progress.
- `registrationPreferences` - optional, an array of properties that can be requested after the user has provided the required properties.

KBA Security Answer Definition Stage

In the context of registration, this stage supplies security questions to the user and captures the answers provided by the user.

The stage validates any answers against the user object. If the requirement is not met (incorrect number of questions answered correctly), the stage throws a bad request exception and increments the failure count of the managed user. If the requirement is met (correct number of questions answered correctly), the process advances to the next stage.

This stage also disallows users from entering custom questions that duplicate any questions defined by the administrator, regardless of the locale. It does this comparison by removing any special characters and making a lowercase comparison. For example, `What Is YoUr FaVorite COLOR????` would be evaluated as the same question as `what is your favorite color?`.

Example configuration

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Dependencies

The stage depends on a previous stage to populate the user ID in `state`. It has no dependencies on following stages.

Required Parameters

- `kbaConfig` - reads the KBA configuration from the corresponding `selfservice.kba.json` file.

KBA Security Answer Verification Stage

This stage verifies security answers and validates user lockout. The stage requires a user ID in `state`.

The stage reads the user object and validates that the user has not already failed to answer the security questions. The stage then obtains the configured security questions, and returns the minimum number of randomly selected questions as a requirement.

The stage validates any answers against the user object. If the requirement is not met (incorrect number of questions answered correctly) the stage throws a bad request exception and increments the failure count of the managed user. If the requirement is met (correct number of questions answered correctly) the process advances to the next stage.

Example configuration

```
{
    "name" : "kbaSecurityAnswerDefinitionStage",
    "kbaConfig" : null
},
```

Dependencies

The stage depends on a previous stage to populate the user ID in `state`. It has no dependencies on following stages.

Required Parameters

- kbaConfig - reads the KBA configuration from the corresponding selfservice.kba.json file.

KBA Update Stage

The KBA Update stage is used as part of progressive profile completion to let users update their existing security questions and to add any additional questions that are needed. This stage updates the user object directly. If a user fails to provide sufficient questions, the stage returns the requirements again. If the object cannot be updated, the stage throws an exception. The stage outputs nothing to the state and has no downstream dependencies.

Example configuration

```
{
    "name": "kbaUpdateStage",
    "kbaConfig": null,
    "identityServiceUrl" : "managed/user",
    "uiConfig" : {
        "displayName" : "Update your security questions",
        "purpose" : "Please review and update your security questions",
        "buttonText" : "Update"
    }
}
```

Dependencies

No dependencies on previous or following stages. This stage can occur anywhere in a process. If the condition evaluates to true, the process moves on to the next stage.

Required Parameters

- kbaConfig - returns the minimum number of security questions that must be provided.
- identityServiceUrl - the managed object type on which this stage acts.
- uiConfig - how the requirements are conveyed to an end user.

Local Auto-Login Stage

This stage is used to perform auto-login with IDM. The stage obtains the OAuth Login from state, and populates the user object (username and password) in state.

The stage adds the OAuth login to the successAdditions (with a value of true) and adds the successURL from its own configuration. If IDM can obtain all those details from state, it takes the user object, locates the username and password, and generates a CREDENTIAL_JWT. That JWT is then placed in the successAdditions parameter.

If IDM is unable to generate the `CREDENTIAL_JWT`, it generates an internal server error (500).

Example configuration

```
{
    "name" : "localAutoLogin",
    "successUrl" : "",
    "identityUsernameField": "userName",
    "identityPasswordField": "password"
}
```

Dependencies

This stage should appear towards the end of a process—it cannot be the first stage in a process.

Required Parameters

- `successURL` - the URL to which an end user should be redirected following successful registration.
- `identityUsernameField` - the managed object property that contains the username.
- `identityPasswordField` - the managed object property that contains the user password.

Parameters Stage

This stage captures parameters in the original request. To advance, the stage assesses the input body. Any values that have been passed in and are listed in the configuration are put into `state`. The stage ignores any values that are not listed in the configuration. The self-service mechanism passes the parameters back to the client at the end of the process.

By default, this stage is required *only* if you are integrating IDM with AM. The stage is added automatically if you use the UI to configure a self-service process, but can generally be ignored unless a custom client or UI requires it.

Example configuration

```
{
    "name" : "parameters",
    "parameterNames" : [
        "returnParams"
    ]
}
```

Dependencies

In all of the default IDM self-service processes, this must be the first stage in the process. In a custom process, the stage has no order dependencies, and can occur anywhere in a process. All this stage does is to copy named parameters into `successAdditions` for the process to output at `tag:end`.

Required Parameters

- parameterNames - a list of parameters the stage supports. These parameters are returned in the requirements.

Patch Object Stage

Currently, this stage is used *only* to patch the managed object with the terms and conditions acceptance obtained from state. If the terms and conditions state is not present, the stage simply advances to the next stage in the process.

Example configuration

```
{
    "name" : "patchObject",
    "identityServiceUrl" : "managed/user"
}
```

Dependencies

This stage requires the Terms and Conditions Stage to have preceded it. It can be followed by any stage and can occur anywhere in a process.

Requirements

- identityServiceUrl - the managed object type on which this stage acts.

Password Reset Stage

This stage updates the managed object directly, changing the value of the configured identityPasswordField. To gather the initial requirements, the stage reads the managed user object, and checks that the email and userID of the object match what is in state. If they do not match, the stage exits with a Bad request exception.

If they do match, the stage returns with its requirements (the new password value). When the requirements are submitted, the stage advances, locates the userId again, and applies the new password. If the password is empty, the stage throws an exception. If the password is valid, the stage patches the managed user object directly to update the password. If the patch fails, the stage returns the requirements again, along with an error message (for example, a password policy requirement).

Example configuration

```
{
    "name" : "resetStage",
    "identityServiceUrl" : "managed/user",
    "identityPasswordField" : "password"
}
```

Dependencies

This stage cannot be the first stage in a process. It expects a previous stage to populate the `userId` and `mail` attributes of the `user` in `state`.

Required Parameters

- `identityServiceUrl` - the managed object type on which this stage acts.
- `identityPasswordField` - the managed object property that contains the user password.

Self-Registration Stage

This is currently the final stage in the default user registration process. The stage obtains all the user details from `state`. When the stage advances, it checks `state` for any `idpdata`, combines that with the user data, and creates the managed user object. This stage *must* occur in any registration process.

Note

If you are integrating IDM with AM, the OpenAM Auto-Login Stage can follow this stage.

Example configuration

```
{
    "name" : "selfRegistration",
    "identityServiceUrl" : "managed/user"
},
```

Dependencies

This stage *must* come after a stage that has populated the user in `state`. If the user is absent, the stage exits with an illegal argument exception.

Required Parameters

- `identityServiceUrl` - the managed object type that the stage creates.

Social User Claim Stage

This stage enables an existing managed user to claim a social identity. The stage obtains a `CLIENT_TOKEN` from some social identity provider. That token includes the following data:

- OAuth token
- Identity provider name
- Renewal token

- Expiration date

Using the CLIENT_TOKEN, the stage retrieves the user profile from the social identity provider and normalizes the profile into a user object (using the regular normalization mapping for social identity providers). For more information on this mapping, see "Many Social Identity Providers, One Schema".

If the stage is unable to retrieve the user profile, or unable to normalize it using the mapping, it exits with an exception. It does not return any missing requirements.

When the user profile has been normalized, the stage attempts to identify any existing managed users that match the profile. If there are no matches, it simply advances to the next stage in the process. If it finds a match, it extracts the existing managed object and returns that as a new set of requirements.

The new requirement is that the user must provide their `password`, either their managed/user password, or the password to another social identity provider, if they registered through a separate identity provider.

The stage then does the following:

- Verifies the login
- Creates a `managed/idp` object for the user
- Establishes a relationship between the managed object and the idp object
- Puts `OAUTH_LOGIN:true` into `state`
- Puts a `claimedProfile` containing the URL of the managed object that was claimed into `successAdditions`

Example configuration

```
{
    "name" : "socialUserClaim",
    "identityServiceUrl" : "managed/user",
    "claimQueryFilter" : "/mail eq \"{{mail}}\""
},
```

Dependencies

This stage has no dependencies on previous or subsequent stages and can occur anywhere in a process.

Required Parameters

- `identityServiceUrl` - the managed object type against which the stage verifies the profile.
- `claimQueryFilter` - the query filter that is used to locate the managed object from the social identity provider profile.

Notice the double-brace notation in preceding example `"claimQueryFilter" : "/mail eq \"{{mail}}\""`. This notation indicates that the named property from the user object in `state` is substituted for the double-braced value. In this example, `{{mail}}` would become the value of the `mail` property of the user in `state`, such as `bjensen@example.com`, if that was in the user in `state`. You can use this notation with any user property.

Terms and Conditions Stage

This stage evaluates a boolean `accepted` (`true` or `false`).

Example configuration

This stage is configured in a `selfservice.terms.json` file in the project `conf` directory and includes the following parameters:

```
{
    "versions" : [
        {
            "version" : "1",
            "termsTranslations" : {
                "en" : "Sample terms and conditions"
            },
            "createDate" : "2018-04-10T09:52:25.478Z"
        }
    ],
    "uiConfig" : {
        "displayName" : "We have updated our terms",
        "purpose" : "To proceed, accept these terms",
        "buttonText" : "Accept"
    },
    "active" : "1"
}
```

The stage can stand on its own (as it does in the default registration configuration) or be called from the Conditional User Stage with a configuration similar to the following:

```
{
    "name" : "conditionaluser",
    "identityServiceUrl" : "managed/user",
    "condition" : {
        "type" : "terms"
    },
    "evaluateConditionOnField" : "user",
    "onConditionTrue" : {
        "name" : "termsAndConditions"
    }
},
```

Dependencies

Configured as part of the Conditional User Stage. *Must* have the Patch Object Stage somewhere downstream. This stage can occur anywhere in a process.

Requirements

Requires Terms and Conditions to be accepted before continuing to the next stage:

- If `accept` is absent, the stage returns the requirements again.
- If `accept` is present but `false`, the stage generates an exception. It is up to the client to handle that exception.
- If `accept` is `true`, this stage puts all the outputs into `state` and advances to the next stage.

Outputs

`TERMS_ACCEPTED`, `TERMS_DATE`, and `TERMS_VERSION`

User Query Stage

This stage queries the managed user repository for a user, based on the supplied query fields. If the stage identifies a user, it populates the `mail`, `userId`, `userName`, and `accountStatus` fields in `state`.

Example configuration

```
{
    "name" : "userQuery",
    "validQueryFields" : [
        "userName",
        "mail",
        "givenName",
        "sn"
    ],
    "identityIdField" : "_id",
    "identityEmailField" : "mail",
    "identityUsernameField" : "userName",
    "identityServiceUrl" : "managed/user",
    "identityAccountStatusField" : "accountStatus"
},
```

Dependencies

This stage has no dependencies on preceding or following stages, but cannot be the only stage in a process.

Required Parameters

- `validQueryFields` - an array of fields on which the query can be based.
- `identityIdField` - the managed object property that contains the user ID to be provided to `state`.
- `identityEmailField` - the managed object property that contains the user mail to be provided to `state`.

- `identityUsernameField` - the managed object property that contains the username to be provided to `state`.
- `identityAccountStatusField` - the managed object property that contains the user account status to be provided to `state`.
- `identityServiceUrl` - the managed object type on which this stage acts.

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.