



Platform Self-Service Guide

/ ForgeRock Identity Platform 7

Latest update: 7.0.0

Copyright © 2020 ForgeRock AS.

Abstract

Guide to setting up User Self-Service in the ForgeRock Identity Platform™.



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.

Table of Contents










Overview	iv
1. User Self-Service Overview	1
Authentication Trees and Self-Service	2
2. Platform Configuration for User Self-Service	5
Configure Self-Service Trees Endpoints	5
Configure Self-Service Policies	6
Configure Email for Self-Service	7
3. User Self-Registration	10
Configure CAPTCHA Services	11
Configure Security Questions	11
Configure Terms and Conditions	13
Configure Privacy and Consent	14
Example Registration REST Output	15
4. Registration using Identity Providers	26
Configure a Basic Social Registration Tree	28
Configure Social Registration with Account Claiming	30
5. Login with Self-Service	32
Configure Login to include Social Identity Providers	33
Example Login REST Output	34
6. Progressive Profile	36
7. Password Reset	38
Example Reset Password REST Output	38
8. Username Recovery	40
Example Forgotten Username REST Output	40
9. Password Updates	42

Overview

The ForgeRock Identity Platform is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

This guide lets you set up and configure user self-service processes using the ForgeRock Identity Platform. These processes include self-registration, login, registration using social identity providers, and additional features like progressive profile completion, password reset, and username recovery.

Quick Start

 <p>Start Here</p> <p>Learn about Self-Service using the ForgeRock Identity Platform.</p>	 <p>Platform Configuration</p> <p>Configure the ForgeRock Identity Platform for Self-Service.</p>	 <p>Self-Registration</p> <p>Configure User Self-Registration.</p>
 <p>Social Registration</p> <p>Configure Registration using external Identity Providers.</p>	 <p>Login</p> <p>Configure the user login flow to use Self-Service.</p>	 <p>Progressive Profile</p> <p>Configure Progressive Profile Completion.</p>
 <p>Password Reset</p> <p>Configure user-driven password reset.</p>	 <p>Username Recovery</p> <p>Configure user-driven username recovery.</p>	 <p>Update Password</p> <p>Configure user-driven password updates.</p>

Note

This guide is meant to help you familiarize yourself with using the platform for user self-service. It will provide examples of common self-service actions and introduce you to core concepts for implementing self-service. However:

- This guide is **not** exhaustive or prescriptive: the examples provided are simply one approach, and there may be more effective approaches for your specific needs. Use the examples as a starting point to learn from.

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>.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1

User Self-Service Overview

User Self-Service lets your users create and manage their own accounts, while giving you control over what features are available and how they work. With the platform, this is done using authentication trees, either through the AM Admin UI, or through the Platform Admin UI. Because this service uses both AM and IDM to work, it requires the platform to function.

Note

While it is possible to configure authentication trees in both the AM Admin UI and the Platform Admin UI, the Platform Admin UI is recommended:

- The Platform Admin UI includes the ability to easily duplicate an existing tree, making it easier to experiment with new flows without changing the behavior of the current tree.
- Some tree-level configuration is not currently available from the AM Admin UI, such as setting the IDM object type you are interacting with, stored in the `identityResource` property in your tree object. This defaults to `managed/user`; to work with a different managed object (`managed/devices`, for example), this will need to be set either through the REST API, or the Platform Admin UI.

One case where you may wish to use the AM Admin UI instead, is to configure trees in different realms.

Before continuing with this guide, make sure you have successfully configured the platform. There are several methods you can use to set up the platform:

- Configure and set up the platform using Kubernetes. More information about setting up the ForgeRock Identity Platform with Kubernetes can be found in the ForgeOps Documentation.
- Alternatively, manually configure the platform integration between AM and IDM. More information is found in "*Deployment Overview*" in the *Platform Setup Guide*.

This guide references some sample authentication trees that have been created to demonstrate various features of self-service. Depending on your configuration method, these trees may already be included. If they aren't already present, or you deleted the trees and wish to re-create them, these sample trees can be found in `sample-trees-7.0.0.zip` included with AM. For more information about adding these trees to the platform, see "Configure Authentication Trees" in the *Platform Setup Guide*.

Note

This guide is focused on the platform implementation of User Self-Service. To use the previous IDM-specific or AM-specific implementations, instructions on how to do so are found in the IDM Self-Service Reference and the AM User Self-Service Guide, respectively.

Authentication Trees and Self-Service

The following nodes were created specifically for use in a platform environment, and are intended for use in self-service flows; you can also use them in other authentication flows in a platform environment:

Nodes Requiring the ForgeRock Identity Platform

Accept Terms and Conditions Node	Attribute Collector Node	Attribute Present Decision Node
Attribute Value Decision Node	Consent Collector Node	Create Object Node
Email Suspend Node	Email Template Node	Identify Existing User Node
Increment Login Count Node	KBA Definition Node	KBA Decision Node
KBA Verification Node	Login Count Decision Node	Patch Object Node
Platform Password Node	Platform Username Node	Profile Completeness Decision Node
Query Filter Decision Node	Required Attributes Present Node	Select Identity Provider Node
Social Provider Handler Node	Terms and Conditions Decision Node	Time Since Decision Node

Since User Self-Service is built using authentication trees, nearly any authentication node included with AM can be used in your self-service flow. The following nodes are *not* compatible with platform-based self-service, however:

Nodes Incompatible with ForgeRock Identity Platform

OAuth 2.0 Node	Social Facebook Node	Social Google Node
Social Ignore Profile Node	OpenID Connect Node	Provision IDM Account Node
Create Password Node	Password Collector Node	Username Collector Node

If you are using a third-party node from the ForgeRock Marketplace, check with the developer for compatibility.

The following sample trees are available:

Registration

The sample Registration tree describes a basic registration flow, where the user is prompted to provide several profile attributes, then attempts to create the user and log the user in. You can find this tree in AM samples in `root/AuthTree/PlatformRegistration.json`. More information is covered in "*User Self-Registration*". For more information about configuring registration to include social identity providers, see "*Registration using Identity Providers*".

Login

The sample Login tree describes a basic login flow, where the user is prompted to provide a username and password, then passed to a progressive profile tree before being logged in. You can find this tree in AM samples in `root/AuthTree/PlatformLogin.json`. More information about modifying the Login tree is covered in "*Login with Self-Service*". For more information about including social identity providers in a Login tree, see "*Registration using Identity Providers*".

Progressive Profiles

The sample Progressive Profile tree is called by the Login tree sample. It checks the login count to see if further action is needed. If no action is required, it returns to the Login tree to complete logging in. If the specified number of logins is reached, it instead checks to see if user preferences have been set, and if not, prompts the user to set those preferences. It then returns to the Login tree to finish logging in. You can find this tree in AM samples in `root/AuthTree/PlatformProgressiveProfile.json`. For more information about using progressive profiling, see "*Progressive Profile*".

Password Reset

The Password Reset sample tree provides a method for users to reset their password by providing their email and answering some security questions. If the questions are answered correctly, the user is emailed a password reset link, which they must click to proceed. They are then presented with a password prompt to enter a new password. You can find this tree in AM samples in `root/AuthTree/PlatformResetPassword.json`. For more information, see "*Password Reset*".

Forgotten Username

The Forgotten Username sample tree gives users a method to recover their username by entering an email address. If the email address is associated with a user account, the account's username will be emailed to the user. The email includes a link to log in, which will take the user through the Login tree. You can find this tree in AM samples in `root/AuthTree/PlatformForgottenUsername.json`. For more information, see "*Username Recovery*".

Update Password

The Update Password sample tree provides a method for users to update their password. This tree assumes the user has already logged in successfully. It checks the user's session data, and if the session is correct, displays a prompt to update the user's password. You can find this tree in AM samples in `root/AuthTree/PlatformUpdatePassword.json`. For more information, see "*Password Updates*".

Note

There is a small naming difference, depending on which method you used to set up the platform. If you are using ForgeOps, the names of the trees will be as listed above. If you manually set up the platform and are loading the trees from the AM samples, the names will have **Platform** prefixed to the tree names (for example, **PlatformRegistration**, or **PlatformForgottenUsername**). The trees and behavior are the same, just with different names.

Chapter 2

Platform Configuration for User Self-Service

Some configuration is necessary to enable Self-Service for the platform. Depending on your method of deployment, some or all of these steps may already be complete, but should be checked to make sure everything is set up correctly.

Most of the configuration for Self-Service is located in AM, with a few exceptions:

- Self-Service Policies
- Email Services and Templates
- Security Questions (KBA)
- Terms & Conditions
- Privacy and Consent

Configure Self-Service Trees Endpoints

AM includes a service to map authentication trees that you created to endpoints in Self-Service. To reach this service, from the top-level realm in AM, go to Services and select the Self Service Trees service. If the service isn't already present, add it using the Add Service button at the top of the page.

You can add multiple endpoints to handle different behavior you want to include. For instance, if you wanted a separate registration flow for registering devices, you could create a tree called `Device Registration`, then add a new endpoint here called `device-registration`, with "Device Registration" as the value.

Note

The `login` endpoint is handled elsewhere. The `login` endpoint is determined by the Organization Authentication Configuration setting for your realm in Authentication > Settings.

To delete an existing endpoint, you need to call this service's endpoint directly, and update the `treeMapping` object:

```
curl \
--request PUT \
--header 'accept: application/json, text/javascript, */*; q=0.01' \
--header 'accept-api-version: protocol=1.0,resource=1.0' \
--header 'accept-language: en-US' \
--header 'content-type: application/json' \
--header 'cookie: <omitted for length>' \
--header 'x-requested-with: XMLHttpRequest' \
--cookie '<omitted for length>' \
--data '{
  "treeMapping":{
    "resetPassword":"PlatformResetPassword",
    "updatePassword":"PlatformUpdatePassword",
    "forgottenUsername":"PlatformForgottenUsername",
    "registration":"PlatformRegistration",
  },
  "_id": "",
  "_type":{
    "_id":"selfServiceTrees",
    "name":"Self Service Trees",
    "collection":false
  }
}' \
https://default.iam.example.com/am/json/realms/root/realm-config/services/selfServiceTrees
```

Configure Self-Service Policies

You can set up policies to determine how different features in Self-Service should behave, such as determining password requirements, or that required fields have been filled out. Policies are configured in IDM. More information about using policies can be found in *Use Policies to Validate Data in the IDM Object Modeling Guide*.

To configure which policies are applied:

1. Open the IDM Admin UI, and select **Configure > Managed Objects**, then select the type of managed object you wish to configure (for example, **User**). This will take you to a list of properties which are part of that object type.
2. Select the property you wish to configure (for example, **password**), then click on the **Validation** tab. This will list any policies currently in place.
3. You can add, remove, or edit policies that are available in IDM. If you need to create a custom policy, see *Extend the Policy Service* in the *IDM Object Modeling Guide*. Please note: creating custom policies is not available through the UI, though they can be set through the IDM Admin UI once the policies have been created.

Note

It's possible to also set password policies within DS. If policies are set in both IDM and DS, make sure the policies match. If the DS password policy is more restrictive than the IDM policy, the user may get an error when updating their password, despite satisfying the policy set in IDM.

Configure Email for Self-Service

The Email Template node and Email Suspend node make use of the email service in IDM. To use email in platform Self-Service, this will need to be configured.

To configure email:

1. Open the IDM Admin UI, then select Configure > Email Settings.
2. If the email service is not yet enabled, select Enable. It will then prompt you to fill out the settings for the email service you intend to use. For more information about configuring email, see [Configure Outbound Email](#) in the *IDM External Services Guide*.
3. Once email service is configured, set up the email templates used in Self-Service by selecting the Templates tab in Email Settings. There are five templates used in default Self-Service flows:
 - **Forgotten Username:** Used in the Forgotten Username tree. When calling this template in a node, the template name is `forgottenUsername`.
 - **Registration:** This template is not used in any of the example trees, but is available if you wish to configure registration to include email verification. When calling this template in a node, the template name is `registration`.
 - **Reset Password:** Used in the Reset Password tree. When calling this template in a node, the template name is `resetPassword`.
 - **Update Password:** This template is not used in any of the example trees, but is available if you wish to configure the Update Password to include an email step. When calling this template in a node, the template name is `updatePassword`.
 - **Welcome:** This template is not used in any of the example trees, but is available if you wish to include a welcome email after the user is registered. When calling this template in a node, the template name is `welcome`.

It is possible to set up additional email templates according to your needs. For example, you may wish to set up an email notification when the user's password is updated. This functionality is not currently available in the UI, however.

To create a new email template:

1. In your IDM `conf/` directory, create a new file called `emailTemplate-newTemplateName.json`. For example, to send a password change notification when a user updates their password, create `emailTemplate-changedPassword.json`.
2. In the new file you created, add the template information. For example, if you wanted to create an `changedPassword` email template:

```
{
  "enabled" : true,
  "from" : "",
  "subject" : {
    "en" : "Password Change Notification"
  },
  "message" : {
    "en" : "<html><body>Your password has just been changed.<br/> If you did not change your
password, or believe you received this email in error, please contact Customer Support.</body></
html>",
  },
  "defaultLocale" : "en",
  "mimeType" : "text/html"
}
```

Note that both `subject` and `message` are localized, and can include HTML tags allowed in HTML emails.

3. Once the template has been added, you can then reference the email template in your Email Template or Email Suspend nodes using the template name (in the above example, `changedPassword`).

The following nodes are associated with platform email services:

Email Suspend Node

The Email Suspend node emails the user using an email template that you have configured in IDM. It then pauses the tree it is used in, until it receives a response from a link the email that was sent. This can be useful in cases of registration, where you wish to include an email verification step, or in a password reset flow, where you want additional verification before proceeding with the password reset.

When using this node, make sure the email template you are using includes a resume link, so the node can continue after the email is received. This is done using the `{{object.resumeURI}}` template variable.

Email Template Node

The Email Template node emails the user using an email template that you have configured in IDM. Unlike the Email Suspend node, this node does not pause the tree. This makes it more useful for cases where you don't need to wait for feedback from the user, such as a Welcome email, or when recovering a username.

There are two possible outcomes: either the email is successfully sent, or the email is not sent. An email might not be sent for different reasons, but most commonly because the email doesn't exist on any known user. For security reasons, we recommend sending both Email Sent and Email Not Sent the same response (Success).

Chapter 3

User Self-Registration

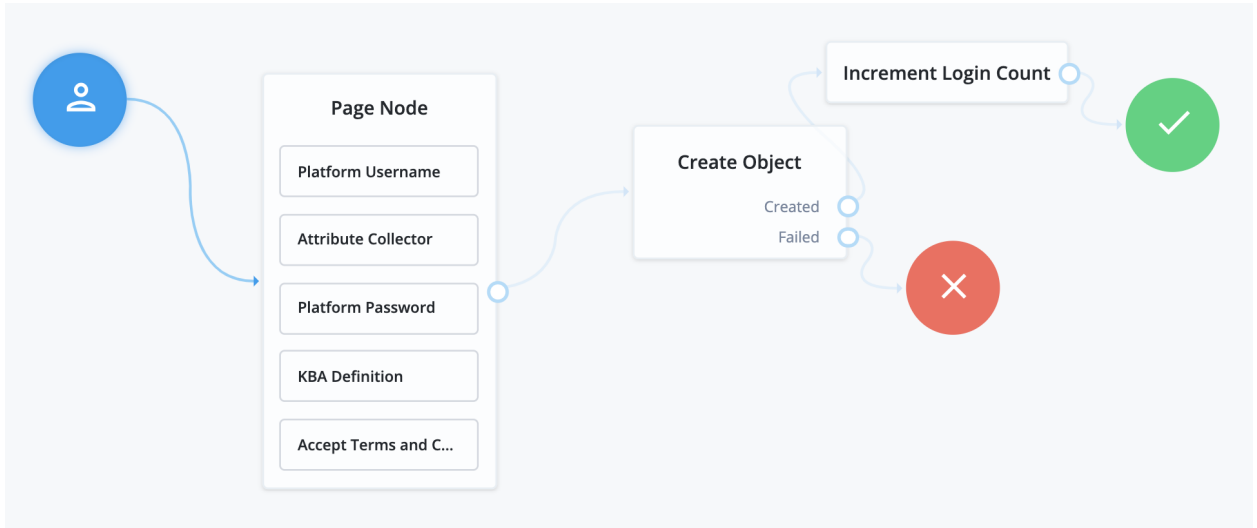
User Self-Registration lets your users create their own accounts. To configure registration, your registration tree requires at least the following nodes:

- The Platform Username node. If you have changed the `userName` attribute to something else, you will need to configure this node to use the new attribute instead (for example, if you changed your configuration to use the `mail` attribute instead).
- An Attribute Collector node, configured to collect information from the user for any attributes that are required to create a new user. By default, required attributes include `userName`, `givenName`, `sn` (short for surname), and `mail` (short for email). The node can collect optional attributes as well, as long as any required attributes are collected.
- The Create Object node, to actually create the user in IDM.

All other nodes are technically optional. Some are strongly encouraged; for example, if you don't include a Platform Password node, the user won't have a password to authenticate and log in with. Cases where the Platform Password node isn't necessary are cases such as if you provide some other method to either authenticate (such as social identity providers), or generate a password for the user.

Nodes that present or collect information for the user are each displayed on their own page by default. To collect multiple nodes into one page, place these nodes in a Page node. There are some limitations to consider when adding nodes to a Page node:

- Only nodes that require interaction with the user should go in a Page node.
- There should be no more than one node with multiple possible outcomes in a Page node.
- The Email Suspend node and the Social Provider Handler node should not be placed in a Page node.



Common nodes to have in a registration tree include:

- The CAPTCHA node, discussed further in CAPTCHA Services.
- The KBA Definition node, discussed further in Security Questions (KBA).
- The Accept Terms and Conditions node, discussed in Terms & Conditions.
- The Consent Collector node, discussed in Privacy and Consent.

Configure CAPTCHA Services

CAPTCHA refers to a way to challenge a user to verify that they are human. A number of CAPTCHA services available. Which you use is up to you - the default configuration in the platform CAPTCHA node are for Google's reCAPTCHA service. The node has been tested for use with reCAPTCHA v2 and hCaptcha v1. Other services should work, as long as they follow a similar configuration pattern.

You will need to provide a CAPTCHA Site Key and CAPTCHA Secret Key. The rest of CAPTCHA configuration is done through the service that you are using.

Configure Security Questions

Security Questions, also known as Knowledge-Based Authentication (KBA), let the user set answers to questions that can be used to verify the user's identity when needed.

Security questions are configured in IDM. In the IDM Admin UI, select Configure > Security Questions. From here, you can configure what questions are available, and how they should be handled:

- Number refers to the number of security questions the user must provide. The minimum is 2, and can be as high as the number of security questions you create.
- Must Answer refers to the number of questions the user must answer to satisfy a security prompt. The minimum number is 1, and can be as high as the amount provided in the Number field.
- Lock Out After refers to the number of failed attempts to answer a security question before the user is unable to try again. Property Name is the name of the property used to store the number of attempts that have been made. By default, these fields are blank; you will need to decide on a property name if you wish to use the lock out functionality.

Note

If you are using an explicit mapping for managed user objects, you must add the property name you set to your database schema *and* to the `objectToColumn` mapping in your repository configuration file.

You also need to create a new column in the `openidm.managed_user` table with the name of your new property, and a datatype of `VARCHAR`.

- Questions lists the currently available security questions. To add a new question, click Add a question. This displays a form, where you can select a locale, and provide the text of a question. When you have added the localized text for your question, click Add, then repeat for each locale. When done with the new question, click Done.

Warning

Once you deploy these security questions, you should never remove or change existing security questions, as users may have included those questions during the user self-registration process.

There are three nodes associated with KBA:

KBA Definition Node

The KBA Definition node is used during registration. It prompts the user to choose security questions, and define answers to these questions for use during identity verification. The questions are selectable from a list. The list also includes an option to define their own question, if they wish.

KBA Verification Node

The KBA Verification node is used to verify a user's identity using security questions, such as during a Reset Password flow. It displays the number of questions set in the Must Answer field in the Security Questions settings. If the user has defined answers for more questions than is required, which questions will be displayed are randomized.

KBA Decision Node

The KBA Decision node is primarily used in cases of a Progressive Profile flow, where you ensure a user has defined answers to the minimum number of questions required by the system. This can be useful if the number of questions changes, so the user can be prompted to fill out any necessary additional questions when they next log in. In this case, the KBA Decision node would be used together with the KBA Definition node; if the KBA Decision node evaluates false, the user would then be taken to the KBA Definition node.

Configure Terms and Conditions

Terms and Conditions display the terms and conditions for using your service. Terms and Conditions are not considered optional; they must accept the terms before they are able to progress in the account creation process.

Terms and Conditions are configured in IDM. To reach these settings, in the IDM Admin UI, select Configure > Terms & Conditions. From here, you can view, set, and add terms and conditions for your service.

To create a new set of terms, click the New Version button. This brings you to the New Terms & Conditions Version page. This page has several fields:

- Version refers to the version of the Terms and Conditions. Terms and Conditions cannot be edited or deleted once removed, and are instead tracked using versioning. The default placeholder set of terms and conditions has a version of **0.0**, but the versioning can follow other patterns, such as the date.
- Make Active determines which version of the terms and conditions is active. Only one version of terms and conditions can be active at a time; setting this will deactivate the currently active version, and make this version active instead.
- Terms & Conditions is the actual text of your terms and conditions. The text supports localization; after creating a locale's terms and conditions, click the Add button. When you are done adding localized versions of the terms, click Save.

Warning

Terms & Conditions versions, once saved, *cannot* be edited, only activated or deactivated. Make sure to proofread your text before saving.

There are two nodes associated with Terms & Conditions:

Accept Terms and Conditions Node

The Accept Terms and Conditions node presents the user with a notice that continuing means they agree with the terms and conditions you have set, along with a link to view the terms and

conditions, and a button to continue. Because this node includes a button to continue by default, it should generally be the last node in a Page node, or on its own page. It will automatically make use of the terms and conditions version that is currently active; you do not need to specify the version in the node.

Terms and Conditions Decision Node

The Terms and Conditions Decision node is used in Progressive Profile trees, where you wish to confirm that the user has accepted the currently active terms and conditions. If the terms and conditions version has been updated, the decision will evaluate to `false`, which, when connected to the Accept Terms and Conditions node, will present the user an opportunity to accept the new terms and conditions.

Configure Privacy and Consent

Privacy and consent, in the context of registration and self-service, refers to presenting users with information about which external resources their information may be shared with, such as sales and marketing services. The ForgeRock Identity Platform manages these connections in IDM, where consent is configured per external connection, or mapping. A mapping refers to the user's information, mapped to related fields in an external service, which is then synchronized by IDM. For more information, see [Mapping Data Between Resources in the IDM *Synchronization Guide*](#).

To enable consent for a mapping:

1. Select Configure > Mappings, then select Edit on the mapping that you wish to configure.
2. Select the Advanced tab, then enable Enable Privacy & Consent.

Note

The above steps assume you have already created at least one mapping. You can also enable Privacy and Consent when creating the mapping: the same Enable Privacy & Consent switch is present when you click Create Mapping during the mapping creation process.

There is one node associated with Privacy and Consent:

Consent Collector Node

The Consent Collector node presents the user with a list of all the mappings the user is affected by that have Privacy and Consent enabled. Each mapping can be individually selected or disabled; if you require all mappings to be allowed, there is an option in the node to make all mappings required.

The node can be used during registration or during progressive profile flows. If using this node in a progressive profile flow, you will need to use the Query Filter Decision node to check for the presence of your desired mappings in the user's `consentedMappings` attribute.

Example Registration REST Output

When calling a registration self-service endpoint, you will receive a JSON object back, containing callbacks for each of the nodes included in the registration tree.

+ *Example based on the sample Registration tree:*

```
{
  "authId": "<omitted for length>",
  "callbacks": [
    {
      "type": "ValidatedCreateUsernameCallback",
      "output": [
        {
          "name": "policies",
          "value": {
            "policyRequirements": [
              "REQUIRED",
              "MIN_LENGTH",
              "VALID_TYPE",
              "UNIQUE",
              "CANNOT_CONTAIN_CHARACTERS"
            ],
            "fallbackPolicies": null,
            "name": "userName",
            "policies": [
              {
                "policyRequirements": [
                  "REQUIRED"
                ],
                "policyId": "required"
              },
              {
                "policyRequirements": [
                  "REQUIRED"
                ],
                "policyId": "not-empty"
              },
              {
                "policyRequirements": [
                  "MIN_LENGTH"
                ],
                "policyId": "minimum-length",
                "params": {
                  "minLength": 1
                }
              }
            ],
            {
              "policyRequirements": [
                "VALID_TYPE"
              ],
              "policyId": "valid-type",
              "params": {
                "types": [
                  "string"
                ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

```

    },
    {
      "policyId": "unique",
      "policyRequirements": [
        "UNIQUE"
      ]
    },
    {
      "policyId": "no-internal-user-conflict",
      "policyRequirements": [
        "UNIQUE"
      ]
    },
    {
      "policyId": "cannot-contain-characters",
      "params": {
        "forbiddenChars": [
          "/"
        ]
      },
      "policyRequirements": [
        "CANNOT_CONTAIN_CHARACTERS"
      ]
    }
  ],
  "conditionalPolicies": null
}
},
{
  "name": "failedPolicies",
  "value": []
},
{
  "name": "validateOnly",
  "value": false
},
{
  "name": "prompt",
  "value": "Username"
}
],
"input": [
  {
    "name": "IDToken1",
    "value": ""
  },
  {
    "name": "IDToken1validateOnly",
    "value": false
  }
],
"_id": 0
},
{
  "type": "StringAttributeInputCallback",
  "output": [
    {
      "name": "name",
      "value": "givenName"
    }
  ]
}

```

```

    },
    {
      "name": "prompt",
      "value": "First Name"
    },
    {
      "name": "required",
      "value": true
    },
    {
      "name": "policies",
      "value": {
        "policyRequirements": [
          "REQUIRED",
          "VALID_TYPE"
        ],
        "fallbackPolicies": null,
        "name": "givenName",
        "policies": [
          {
            "policyRequirements": [
              "REQUIRED"
            ],
            "policyId": "required"
          },
          {
            "policyRequirements": [
              "VALID_TYPE"
            ],
            "policyId": "valid-type",
            "params": {
              "types": [
                "string"
              ]
            }
          }
        ]
      },
      "conditionalPolicies": null
    }
  ],
  {
    "name": "failedPolicies",
    "value": []
  },
  {
    "name": "validateOnly",
    "value": false
  },
  {
    "name": "value",
    "value": ""
  }
],
"input": [
  {
    "name": "IDToken2",
    "value": ""
  },
  {

```

```

    "name": "IDToken2validateOnly",
    "value": false
  }
],
"_id": 1
},
{
  "type": "StringAttributeInputCallback",
  "output": [
    {
      "name": "name",
      "value": "sn"
    },
    {
      "name": "prompt",
      "value": "Last Name"
    },
    {
      "name": "required",
      "value": true
    },
    {
      "name": "policies",
      "value": {
        "policyRequirements": [
          "REQUIRED",
          "VALID_TYPE"
        ],
        "fallbackPolicies": null,
        "name": "sn",
        "policies": [
          {
            "policyRequirements": [
              "REQUIRED"
            ],
            "policyId": "required"
          },
          {
            "policyRequirements": [
              "VALID_TYPE"
            ],
            "policyId": "valid-type",
            "params": {
              "types": [
                "string"
              ]
            }
          }
        ]
      }
    },
    {
      "name": "conditionalPolicies": null
    }
  ],
  "name": "failedPolicies",
  "value": []
},
{
  "name": "validateOnly",
  "value": false
}

```

```

    },
    {
      "name": "value",
      "value": ""
    }
  ],
  "input": [
    {
      "name": "IDToken3",
      "value": ""
    },
    {
      "name": "IDToken3validateOnly",
      "value": false
    }
  ],
  "_id": 2
},
{
  "type": "StringAttributeInputCallback",
  "output": [
    {
      "name": "name",
      "value": "mail"
    },
    {
      "name": "prompt",
      "value": "Email Address"
    },
    {
      "name": "required",
      "value": true
    },
    {
      "name": "policies",
      "value": {
        "policyRequirements": [
          "REQUIRED",
          "VALID_TYPE",
          "VALID_EMAIL_ADDRESS_FORMAT"
        ],
        "fallbackPolicies": null,
        "name": "mail",
        "policies": [
          {
            "policyRequirements": [
              "REQUIRED"
            ],
            "policyId": "required"
          },
          {
            "policyRequirements": [
              "VALID_TYPE"
            ],
            "policyId": "valid-type",
            "params": {
              "types": [
                "string"
              ]
            }
          }
        ]
      }
    }
  ]
}

```



```

    }
    },
    {
      "policyId": "valid-email-address-format",
      "policyRequirements": [
        "VALID_EMAIL_ADDRESS_FORMAT"
      ]
    }
  ],
  "conditionalPolicies": null
}
},
{
  "name": "failedPolicies",
  "value": []
},
{
  "name": "validateOnly",
  "value": false
},
{
  "name": "value",
  "value": ""
}
],
"input": [
  {
    "name": "IDToken4",
    "value": ""
  },
  {
    "name": "IDToken4validateOnly",
    "value": false
  }
],
"_id": 3
},
{
  "type": "BooleanAttributeInputCallback",
  "output": [
    {
      "name": "name",
      "value": "preferences/marketing"
    },
    {
      "name": "prompt",
      "value": "Send me special offers and services"
    },
    {
      "name": "required",
      "value": true
    },
    {
      "name": "policies",
      "value": {}
    },
    {
      "name": "failedPolicies",
      "value": []
    }
  ]
}

```

```

    },
    {
      "name": "validateOnly",
      "value": false
    },
    {
      "name": "value",
      "value": false
    }
  ],
  "input": [
    {
      "name": "IDToken5",
      "value": false
    },
    {
      "name": "IDToken5validateOnly",
      "value": false
    }
  ],
  "_id": 4
},
{
  "type": "BooleanAttributeInputCallback",
  "output": [
    {
      "name": "name",
      "value": "preferences/updates"
    },
    {
      "name": "prompt",
      "value": "Send me news and updates"
    },
    {
      "name": "required",
      "value": true
    },
    {
      "name": "policies",
      "value": {}
    },
    {
      "name": "failedPolicies",
      "value": []
    },
    {
      "name": "validateOnly",
      "value": false
    },
    {
      "name": "value",
      "value": false
    }
  ],
  "input": [
    {
      "name": "IDToken6",
      "value": false
    },
  ],

```

```

    {
      "name": "IDToken6validateOnly",
      "value": false
    }
  ],
  "_id": 5
},
{
  "type": "ValidatedCreatePasswordCallback",
  "output": [
    {
      "name": "echo0n",
      "value": false
    },
    {
      "name": "policies",
      "value": {
        "policyRequirements": [
          "REQUIRED",
          "MIN_LENGTH",
          "VALID_TYPE",
          "AT_LEAST_X_CAPITAL_LETTERS",
          "AT_LEAST_X_NUMBERS",
          "CANNOT_CONTAIN_OTHERS"
        ],
        "fallbackPolicies": null,
        "name": "password",
        "policies": [
          {
            "policyRequirements": [
              "REQUIRED"
            ],
            "policyId": "not-empty"
          },
          {
            "policyRequirements": [
              "MIN_LENGTH"
            ],
            "policyId": "minimum-length",
            "params": {
              "minLength": 8
            }
          },
          {
            "policyRequirements": [
              "VALID_TYPE"
            ],
            "policyId": "valid-type",
            "params": {
              "types": [
                "string"
              ]
            }
          },
          {
            "policyId": "at-least-X-capitals",
            "params": {
              "numCaps": 1
            }
          }
        ]
      }
    }
  ]
}

```

```

        "policyRequirements": [
          "AT_LEAST_X_CAPITAL_LETTERS"
        ]
      },
      {
        "policyId": "at-least-X-numbers",
        "params": {
          "numNums": 1
        },
        "policyRequirements": [
          "AT_LEAST_X_NUMBERS"
        ]
      },
      {
        "policyId": "cannot-contain-others",
        "params": {
          "disallowedFields": [
            "userName",
            "givenName",
            "sn"
          ]
        },
        "policyRequirements": [
          "CANNOT_CONTAIN_OTHERS"
        ]
      }
    ],
    "conditionalPolicies": null
  }
},
{
  "name": "failedPolicies",
  "value": []
},
{
  "name": "validateOnly",
  "value": false
},
{
  "name": "prompt",
  "value": "Password"
}
],
"input": [
  {
    "name": "IDToken7",
    "value": ""
  },
  {
    "name": "IDToken7validateOnly",
    "value": false
  }
],
"_id": 6
},
{
  "type": "KbaCreateCallback",
  "output": [
    {

```

```

    "name": "prompt",
    "value": "Select a security question"
  },
  {
    "name": "predefinedQuestions",
    "value": [
      "What's your favorite color?",
      "Who was your first employer?"
    ]
  }
],
"input": [
  {
    "name": "IDToken8question",
    "value": ""
  },
  {
    "name": "IDToken8answer",
    "value": ""
  }
],
"_id": 7
},
{
  "type": "KbaCreateCallback",
  "output": [
    {
      "name": "prompt",
      "value": "Select a security question"
    },
    {
      "name": "predefinedQuestions",
      "value": [
        "What's your favorite color?",
        "Who was your first employer?"
      ]
    }
  ],
  "input": [
    {
      "name": "IDToken9question",
      "value": ""
    },
    {
      "name": "IDToken9answer",
      "value": ""
    }
  ],
  "_id": 8
},
{
  "type": "TermsAndConditionsCallback",
  "output": [
    {
      "name": "version",
      "value": "0.0"
    },
    {
      "name": "terms",

```

```
"value": "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."
  },
  {
    "name": "createDate",
    "value": "2019-10-28T04:20:11.320Z"
  }
],
"input": [
  {
    "name": "IDToken10",
    "value": false
  }
],
"_id": 9
}
],
"header": "Sign Up",
"description": "Signing up is fast and easy.<br>Already have an account? <a href='#/service/Login'>Sign In</a>"
}
```

Chapter 4

Registration using Identity Providers

You can configure registration to include social identity providers (IdP) as an option for users. This lets users register and log in to your service using an account they have through another trusted service, such as Google, LinkedIn, or Twitter.

Configure Social Identity Providers

1. First enable the Social Identity Provider Service in AM. To do so, in the AM Admin UI, select Configure > Global Services, then select Social Identity Provider Service from the list of services. Turn the Enabled switch on, if it isn't on already.
2. Open the realm you wish to enable identity providers for in the AM Admin UI. Select Services, then select Social Identity Provider Service from the list of services, and make sure the Enabled switch is on in the Configuration tab.
3. ForgeRock Identity Platform includes scripts and configurations for several common identity providers. To configure a social identity provider, select the Secondary Configurations tab, then select the identity provider you wish to configure. If you don't see the service you wish to include in the list, you may be able to add the service manually. For more information about manually adding a new identity provider, see "Add a New Identity Provider".
4. Once you have selected an identity provider, you will need to configure it with the settings given to you by the provider. Most fields are preconfigured for each provider, but you should double check with the provider to make sure those settings haven't changed. You will need to add your own Client ID and Client Secret, which are generated by the provider service.

Also double-check the Transform Script set at the bottom of the configuration page. These scripts are written either in Groovy or JavaScript, and work to translate the attributes received from the identity provider into attributes the ForgeRock Identity Platform can use. You can view or edit these scripts by selecting Scripts in the AM Admin UI sidebar for your chosen realm.

5. Once the identity provider configuration is complete, turn the provider on by selecting the Enabled switch at the top of the provider configuration page, then select Save Changes at the bottom of the page. Only the specific providers you have configured and enabled will be available to users.

If you need support for a social identity provider that is not available by default in the ForgeRock Identity Platform, you can manually add new providers, as long as they have a solution implemented using either OAuth 2.0, or OpenID Connect.

Add a New Identity Provider

1. Open the realm you wish to enable identity providers for in the AM Admin UI. Select Services, then select Social Identity Provider Service from the list of services, then click the Secondary Configuration tab.
2. Select Add a Secondary Configuration, and choose either Client Configuration for providers that implement the OAuth2 specification, or Client Configuration for providers that implement the OpenID Connect specification, depending on what your new identity provider uses.
3. The configuration details will need to be provided by the identity provider in question. Most fields are required (Scope Delimiter is frequently just a space, so may *look* blank, but isn't).
4. The UI Config Properties section of the configuration will depend on which UI you plan to use, and how you want your identity providers to be displayed. If you are using the ForgeRock Identity Platform End User UI, some common properties include:
 - **buttonImage**: A relative path to an image in the End User UI.
 - **buttonCustomStyle**: Any custom CSS you wish to apply to the button outside of normal End User UI styling.
 - **buttonClass**: Adds the specified class to the identity provider button, for any additional styling you want to apply.
 - **buttonCustomStyleHover**: Adds custom styling when the cursor is hovering over the button.
 - **buttonDisplayName**: The name of the identity provider, which will be included either on the button or in the button's `alt` attribute, depending on styling.
 - **iconFontColor**: Specifies the color of the icon. You can use methods supported in CSS (such as `white`, or `#ffffff`).
 - **iconClass**: Adds the specified class to the identity provider icon, for any additional styling you want to apply.
 - **iconBackground**: The color for the background of the icon. You can use methods supported in CSS (such as `white`, or `#ffffff`).
5. You need to create a new Social Identity Provider Profile Transformation script. This script adapts the fields received by the provider to align with the fields expected by the platform. You can write this script using either Groovy or JavaScript; the default scripts included with the platform are written in Groovy. Look at the the Google Profile Normalization script as an example:


```
import static org.forgerock.json.JsonValue.field
import static org.forgerock.json.JsonValue.json
import static org.forgerock.json.JsonValue.object

return json(object(
    field("id", rawProfile.sub),
    field("displayName", rawProfile.name),
    field("givenName", rawProfile.given_name),
    field("familyName", rawProfile.family_name),
    field("photoUrl", rawProfile.picture),
    field("email", rawProfile.email),
    field("username", rawProfile.email),
    field("locale", rawProfile.locale)))
```

In this example, you'll see the script returns a JSON object, where each field is mapped to a normalized version of the field. The first field (for instance, `id`) is the platform attribute name, while the second (for instance, `rawProfile.sub`) is the field received from the provider. Note that some field names are the same (`email` and `rawProfile.email`, for example). These fields still need to be mapped, so they are included in the returned JSON object.

There are two nodes associated with Identity Providers:

Select Identity Provider Node

The Select Identity Provider node is used both during registration flows and during login flows. It prompts the user to select a social identity provider to register or log in with, or (optionally) continue on with a local registration or login flow. When a provider is selected, the flow continues on to the Social Provider Handler node.

Social Provider Handler Node

The Social Provider Handler node is used in combination with the Select Identity Provider node. It communicates with the selected provider and then collects the information provided after the user has authorized the service. It then takes that information and runs a transformation script to prepare it for the platform. The ForgeRock Identity Platform includes a transformation script called Normalized Profile to Managed User, which is used by this node. To add this data to a different managed object, you will need to create a new script customized to the needs of that object.

The node then queries IDM to see if the user already exists. If the user exists, they are logged in. If the user does not exist, the user will need to be created.

Configure a Basic Social Registration Tree

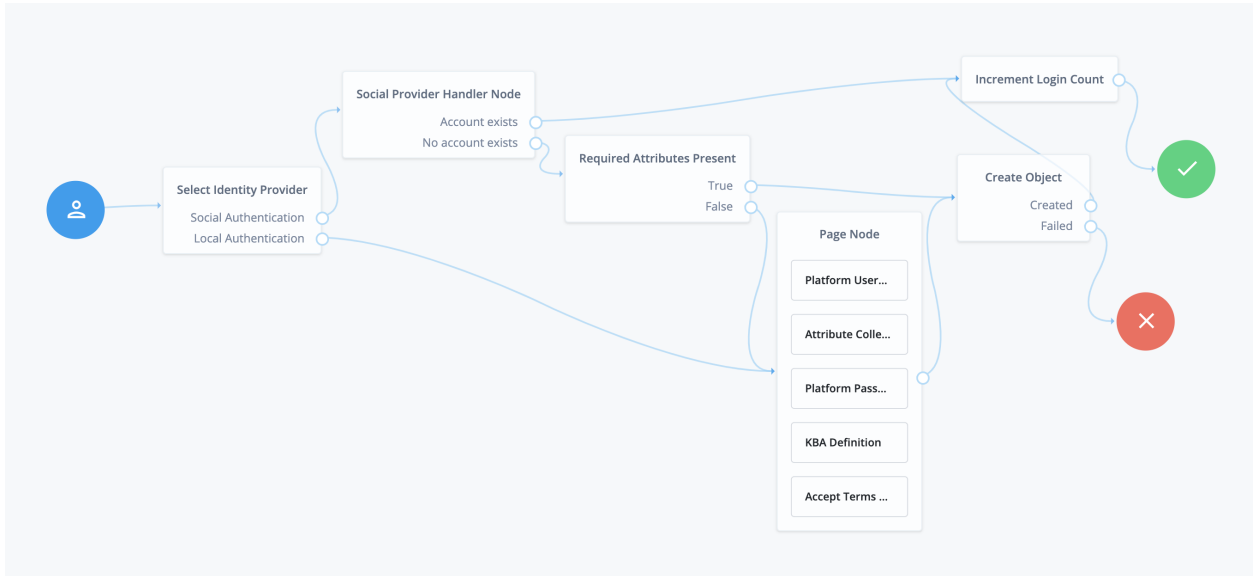
1. To get started with social registration, you can create a new tree, modify the existing registration tree, or duplicate the registration tree and modify that.

+ *This example will use the following nodes:*

- A Page node with the following nodes inside:
 - A Platform Username node.
 - An Attribute Collector node, configured to collect (at least) the following attributes: `givenName`, `sn`, and `mail`.
 - A Platform Password node.
 - A KBA Definition node.
 - An Accept Terms and Conditions node.
- A Create Object node.
- An Increment Login Count node.
- A Select Identity Provider node.
- A Social Provider Handler node.
- A Required Attributes Present node.

2. From the starting User node, connect to the Select Identity Provider node. If it isn't already enabled, turn on Include Local Authentication.
3. Connect the Local Authentication outlet to the Page node that contains the Platform Username, Platform Password, and Attribute Collector nodes.
4. Connect the Page node mentioned above to the Create Object node.
5. Connect the Success output of the Create Object node to the Increment Login Count node. The Increment Login Count node should then be connected to the Success node (or a Success URL node). Connect the Failure output of the Create Object node to the Failure node.
6. Go back to the Select Identity Provider node, this time connecting the Social Authentication to the Social Provider Handler node.
7. From the Social Provider Handler node, connect the Account Exists output to the Increment Login Count node. Connect the No Account Exists output to the Required Attributes Present node.
8. From the Required Attributes Present node, connect the True output to the Create Object node. Connect the False output to the Page node where attributes are collected.

The resulting registration tree will look something like this:

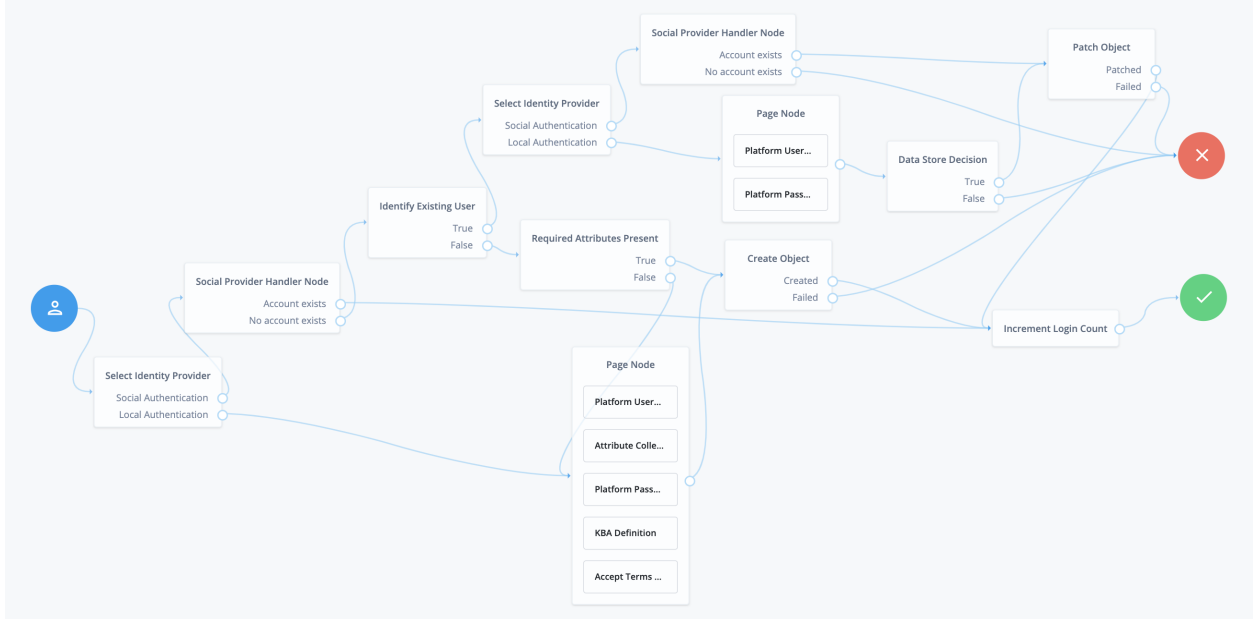


Configure Social Registration with Account Claiming

- Starting with the Social Registration tree created in the previous section, add the following nodes:
 - An Identify Existing User node.
 - A second Select Identity Provider node.
 - A second Social Provider Handler node.
 - A second Page node, containing a Platform Username node and a Platform Password node.
 - A Patch Object node.
 - A Data Store Decision node.
- From the first Social Provider Handler node, connect the No account exists output to the Identify Existing User node. Connect the False output on the Identify Existing User node to the Required Attributes Present node.
- From the Identify Existing User node, connect the True output to the second Select Identity Provider node. In the settings for the Select Identity Provider node, enable Offer only existing providers.

4. From the second Select Identity Provider node, connect the Social Authentication output to the second Social Provider Handler node.
5. From the second Social Provider Handler node, connect the Account Exists output to the Patch Object node. Connect the No Account Exists output to the Failure node.
6. From the Patch Object node, connect the Patched output to the Increment Login Count node. Connect the Failed output to the Failure node.
7. Returning to the second Select Identity Provider node, connect the Local Authentication output to the second Page node (the one containing only a Platform Username node and Platform Password node).
8. Connect the second Page node to the Data Store Decision node. Connect the True output to the Patch Object node. Connect the False output to the Failure node.

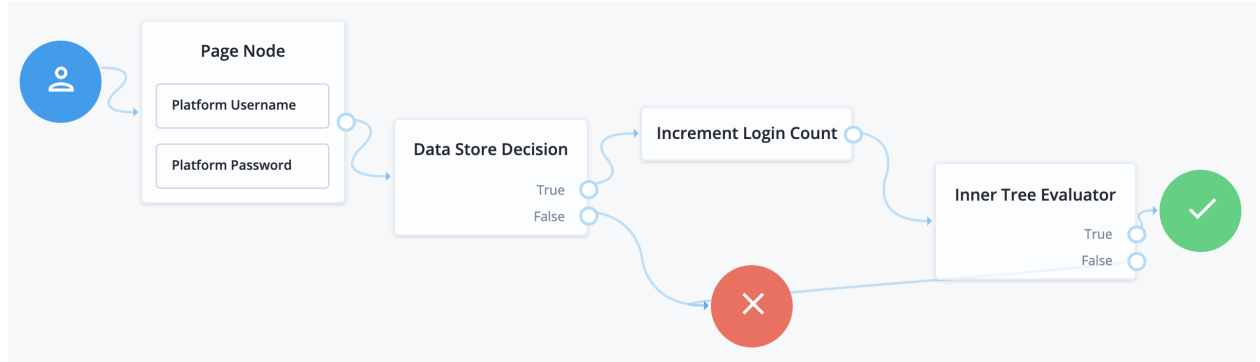
The resulting tree will look something like this:



Chapter 5

Login with Self-Service

The ForgeRock Identity Platform login flow is set up to use self-service, which can be seen in the sample Login authentication tree. This tree lets users log in using their platform credentials and increment a login counter. Users are then sent through a separate Progressive Profile tree. The Login tree can be expanded to include other features, such as support for Identity Providers. For more information about adding support for Identity Providers, see "[Registration using Identity Providers](#)".



The following nodes are associated with Login trees:

Platform Username Node

The Platform Username node is used in both Login and Registration trees. It collects the username of the user. This is similar in behavior to the Username Collector node, but is designed to work in an integrated platform environment.

Platform Password Node

The Platform Password node is used in both Login and Registration trees. It collects the password of the user. This is similar in behavior to the Password Collector node, but is designed to work in an integrated platform environment.

Data Store Decision Node

The Data Store Decision node takes a username and password and validates they match an existing user in the configured data store (in this case, an IDM managed user). This node is not exclusive to a platform environment.

Configure Login to include Social Identity Providers

To include social identity providers as a method of authentication, you will need to enable the Social Identity Provider Service in AM, and include either some form of social registration or social account claiming. For more information about configuring the platform for identity providers, see "*Registration using Identity Providers*". Once this is set up, you will need to add social identity provider support to your Login tree.

1. To get started with social logins, you can create a new tree, modify the existing login tree, or duplicate the login tree and modify that.

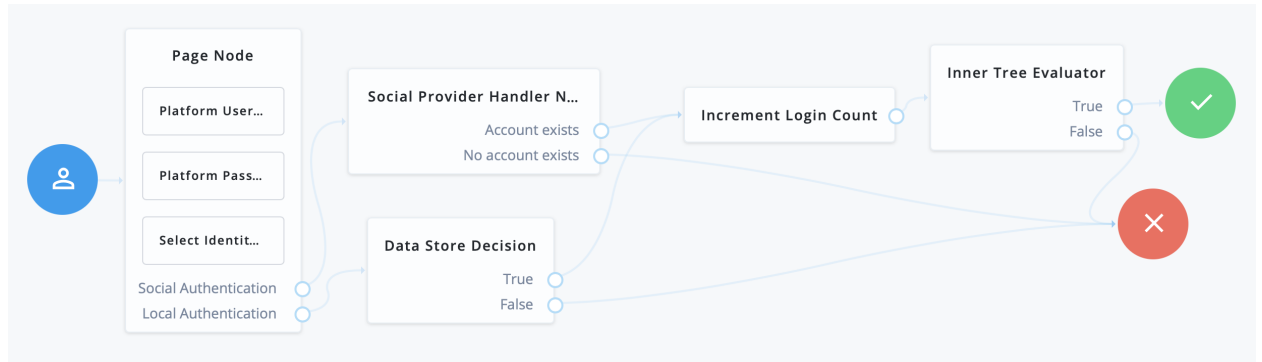
+ *This example will use the following nodes:*

- A Page node containing:
 - A Platform Username node.
 - A Platform Password node.
 - A Select Identity Provider node.
- A Social Provider Handler node.
- A Data Store Decision node.
- An Increment Login Count node.
- An Inner Tree Evaluator node.

2. Connect the starting User node to the Page node.
3. Connect the Social Authentication output on the Page node to the Social Provider Handler node.
4. On the Social Provider Handler node, connect the Account Exists output to the Increment Login Count node. Connect the No Account Exists output to the Failure node.
5. On the Page node, connect the Local Authentication node to the Data Store Decision node.
6. On the Data Store Decision node, connect the True output to the Increment Login Count node. Connect the False output to the Failure node.
7. Connect the Increment Login Count node to the Inner Tree Evaluator node.
8. The Inner Tree Evaluator node points to another tree, letting you chain multiple trees together. By default, this is set to point to the `ProgressiveProfile` tree. For more information about Progressive Profiles, see "*Progressive Profile*".

Connect the Inner Tree Evaluator node to the Success node.

The resulting login tree will look something like this:



Example Login REST Output

When calling a login self-service endpoint, you will receive a JSON object back, containing callbacks for each of the nodes included in the Login tree.

+ *Example based on the sample Login tree:*

```
{
  "authId": "<omitted for length>",
  "callbacks": [
    {
      "type": "ValidatedCreateUsernameCallback",
      "output": [
        {
          "name": "policies",
          "value": {}
        },
        {
          "name": "failedPolicies",
          "value": []
        },
        {
          "name": "validateOnly",
          "value": false
        },
        {
          "name": "prompt",
          "value": "Username"
        }
      ],
      "input": [
        {
          "name": "IDToken1",
          "value": ""
        },
        {
          "name": "IDToken1validateOnly",

```

```

    "value": false
  }
],
"_id": 0
},
{
  "type": "ValidatedCreatePasswordCallback",
  "output": [
    {
      "name": "echo0n",
      "value": false
    },
    {
      "name": "policies",
      "value": {}
    },
    {
      "name": "failedPolicies",
      "value": []
    },
    {
      "name": "validate0nly",
      "value": false
    },
    {
      "name": "prompt",
      "value": "Password"
    }
  ],
  "input": [
    {
      "name": "IDToken2",
      "value": ""
    },
    {
      "name": "IDToken2validate0nly",
      "value": false
    }
  ],
  "_id": 1
}
],
"header": "Sign In",
"description": "New here? <a href=\"#/service/Registration\">Create an account</a><br><a href=\"#/service/ForgottenUsername\">Forgot username?</a><a href=\"#/service/ResetPassword\"> Forgot password?</a>"
}

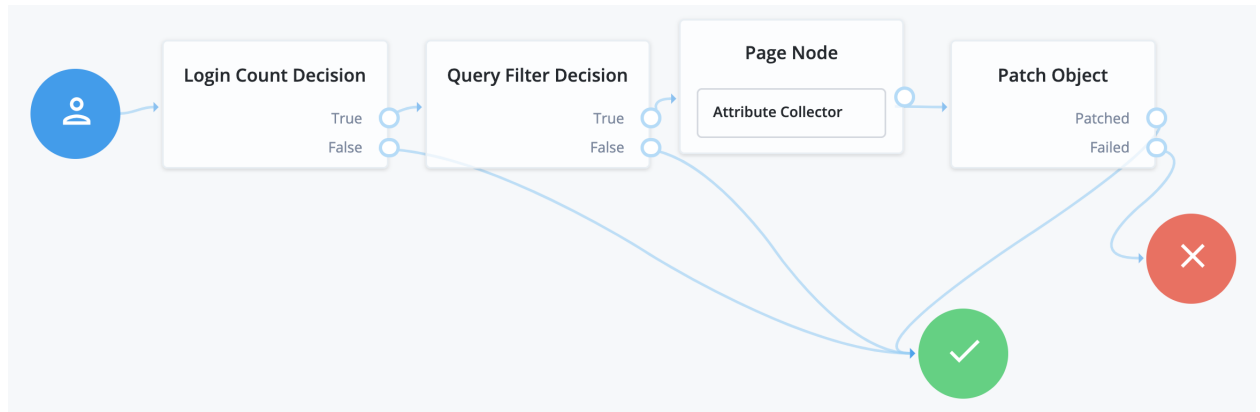
```


Chapter 6

Progressive Profile

Progressive Profile refers to the ability to ask users to provide additional profile information over time, or to update existing information when needed. The sample Progressive Profile tree checks the number of logins, and prompts the user to fill out their marketing preferences if they haven't already. There are a wide variety of other ways you can configure a progressive profile flow, however.

Progressive Profile trees generally aren't linked to directly. Instead, they are included inside other trees, using the Inner Tree Evaluator node. You can connect multiple Inner Tree Evaluator nodes together, which can help you keep different progressive profile behavior organized into their own trees.



The following nodes are associated with progressive profiles:

Attribute Present Decision Node

The Attribute Present Decision node checks to see if the specified attribute is present. It does not check the value of the attribute, only that the attribute exists. This can include attributes that might otherwise be private. A common use case for this node is when you wish to check for the presence of a password.

Attribute Value Decision Node

The Attribute Value Decision node checks the value of the specified attribute, and determines if it satisfies the conditions configured in the node. It can perform two types of comparison operations: it can check whether an attribute is present, or it can check if the value of an attribute equals a value specified in the node.

Like the Attribute Present node, one of the possible conditions you can set is whether an attribute is present. Unlike the Attribute Present node, this will not work on private attributes.

KBA Decision Node

The KBA Decision node is primarily used in cases of a Progressive Profile flow, where you wish to ensure a user has defined answers to the minimum number of questions required by the system. This can be useful if the number of questions changes, so the user can be prompted to fill out any necessary additional questions when they next log in. In this case, the KBA Decision node would be used together with the KBA Definition node: if the KBA Decision node evaluates false, the user would then be taken to the KBA Definition node.

Login Count Decision Node

The Login Count Decision node checks to see if the user has logged in the specified number of times. It can either be triggered once (using the **AT** interval), or triggered repeatedly after a set number of logins (using the **EVERY** interval). The login count is not automatically incremented: be sure to include the Increment Login Count node in your Login tree if you plan to use this node.

Profile Completeness Decision Node

The Profile Completeness Decision node checks how complete a user's profile is, and compares that amount with a percentage value set in the node. The value for profile completeness is based on the number of visible, user-editable attributes in their profile that have been filled out.

Query Filter Decision Node

The Query Filter Decision node uses a query filter to check a user's profile for specific information. Use this to check whether a particular field has been filled out, or that the contents of a field match a specific pattern. For instance, you can use this in progressive profile flows to check if marketing preferences are set on a user's profile. For more information on constructing effective query filters, see *Construct Queries* in the *IDM Object Modeling Guide*.

Terms and Conditions Decision Node

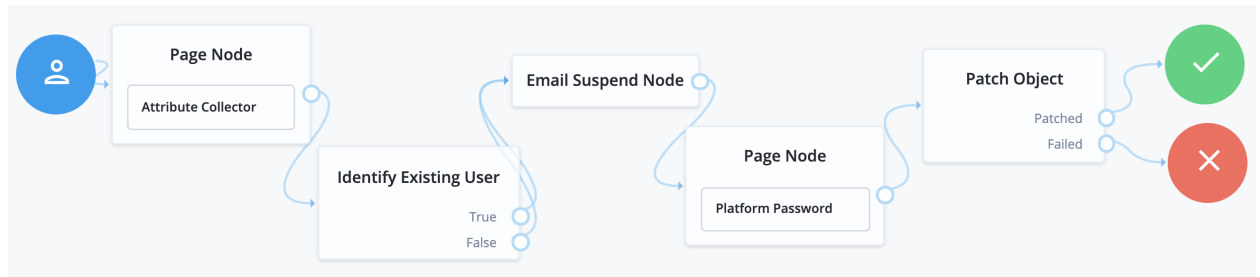
The Terms and Conditions Decision node verifies the user has accepted the currently active set of Terms and Conditions. Use this node when you want to verify the user has accepted your current terms and conditions before proceeding. Use this with the Accept Terms and Conditions node: connect the Terms and Conditions Decision node False output to an Accept Terms and Conditions node.

Time Since Decision Node

The Time Since Decision node checks the user's creation date against a specified amount of time. This is used when you want to have a time-based reminder for users to check an attribute. Once the specified amount of time has elapsed, the node will evaluate to **True** the next time the node is triggered (such as by the user logging in and going through a progressive profile tree).

Chapter 7 Password Reset

Password Reset lets users reset their password without assistance from an administrator. The ForgeRock Identity Platform includes a sample Reset Password tree, which requests a user's email address, checks if a user with that email exists, and if so, emails a reset link to the user. The tree then waits until the user clicks the link before presenting a password reset prompt.



Example Reset Password REST Output

When calling a reset password self-service endpoint, you will receive a JSON object back, containing callbacks for each of the nodes included in the reset password tree.

+ *Example based on the sample Reset Password tree:*

```
{
  "authId": "<omitted for length>",
  "callbacks": [
    {
      "type": "StringAttributeInputCallback",
      "output": [
        {
          "name": "name",
          "value": "mail"
        },
        {
          "name": "prompt",
          "value": "Email Address"
        },
        {
          "name": "required",
          "value": true
        }
      ]
    }
  ]
}
```

```
    "name": "policies",
    "value": {}
  },
  {
    "name": "failedPolicies",
    "value": []
  },
  {
    "name": "validateOnly",
    "value": false
  },
  {
    "name": "value",
    "value": ""
  }
],
"input": [
  {
    "name": "IDToken1",
    "value": ""
  },
  {
    "name": "IDToken1validateOnly",
    "value": false
  }
],
"_id": 0
}
],
"header": "Reset Password",
"description": "Enter your email address or <a href=\\"#/#/service/Login\\">Sign in</a>"
}
```

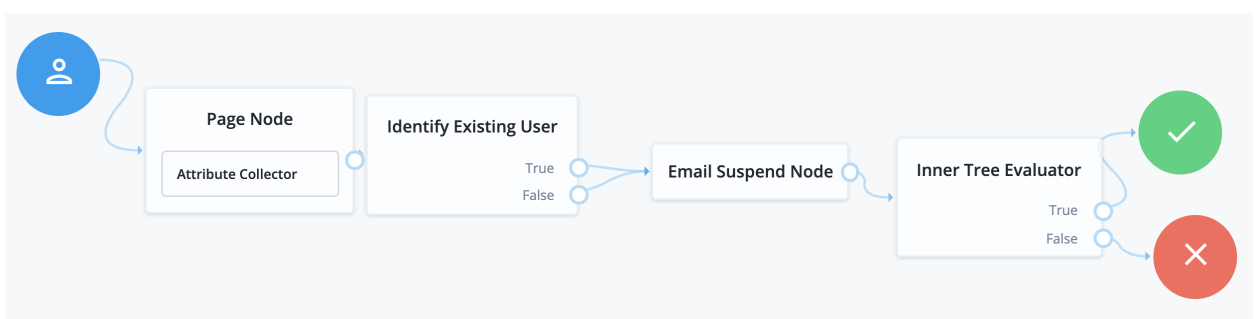
Chapter 8

Username Recovery

Username Recovery lets the user recover their username, using other information they do remember, such as their email address. The ForgeRock Identity Platform includes a sample Forgotten Username tree that is used for this purpose. It collects a user's email address, then uses that to search for a user with that address. It then emails the user the username associated with that email address. An alternative flow for this tree is to send a verification link, then use the Display Username node once the user returns from the email.

Note

When reviewing the example tree, you can see both of the outputs Identify Existing User node connect to the Email Suspend node. This is recommended behavior for security reasons; if you return different outcomes, you can potentially expose which users have accounts in your system.



Example Forgotten Username REST Output

When calling a username recovery self-service endpoint, you will receive a JSON object back, containing callbacks for each of the nodes included in the username recovery tree.

+ *Example based on the sample Forgotten Username tree:*

```
{
  "authId": "<omitted for length>",
  "callbacks": [
    {
      "type": "StringAttributeInputCallback",
      "output": [
```

```
{
  {
    "name": "name",
    "value": "mail"
  },
  {
    "name": "prompt",
    "value": "Email Address"
  },
  {
    "name": "required",
    "value": true
  },
  {
    "name": "policies",
    "value": {}
  },
  {
    "name": "failedPolicies",
    "value": []
  },
  {
    "name": "validateOnly",
    "value": false
  },
  {
    "name": "value",
    "value": ""
  }
},
"input": [
  {
    "name": "IDToken1",
    "value": ""
  },
  {
    "name": "IDToken1validateOnly",
    "value": false
  }
],
"_id": 0
}
],
"header": "Forgotten Username",
"description": "Enter your email address or <a href=\"#/service/Login\">Sign in</a>"
}
```

Chapter 9

Password Updates

Password Updates provide a method for the user to update their password without assistance from an administrator. The ForgeRock Identity Platform includes a sample Update Password tree. Unlike the other sample self-service trees, the Update Password tree assumes the user is already logged in, and gets the user's current session data to identify the user. It then presents a prompt to update the user's password, and uses a Patch Object node to update the password in IDM. An example of where you might use a tree like this is in an Update Password link placed in the user's profile or settings.

