



Platform Self-Service Guide

/ ForgeRock Identity Platform 7.1

Latest update: 7.1.0

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

Copyright © 2021 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.

© Copyright 2010-2020 ForgeRock, Inc. All rights reserved. ForgeRock is a registered trademark of ForgeRock, Inc. Other marks appearing herein may be trademarks of their respective owners.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, and distribution. No part of this product or document may be reproduced in any form by any means without prior written authorization of ForgeRock and its licensors, if any.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESSED OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

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 NON-INFRINGEMENT 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 NON-INFRINGEMENT 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 at 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	v
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. Social Authentication	26
Configure Social Identity Providers	26
Configure a Basic Social Registration Tree	46
Configure Social Registration with Account Claiming	48
5. Login with Self-Service	50
Configure Login to include Social Identity Providers	51
Example Login REST Output	52
6. Progressive Profile	54
7. Password Reset	56
Example Reset Password REST Output	56
8. Username Recovery	58
Example Forgotten Username REST Output	58
9. Password Updates	60
10. Platform Authentication Nodes Reference	61
Accept Terms and Conditions Node	61
Attribute Collector Node	62
Attribute Present Decision Node	63
Attribute Value Decision Node	63
Consent Collector Node	64
Create Object Node	64
Email Suspend Node	65
Email Template Node	66
Identify Existing User Node	67
Increment Login Count Node	68
KBA Decision Node	68
KBA Definition Node	68
KBA Verification Node	69
Login Count Decision Node	69
Page Node	70
Patch Object Node	74
Platform Password Node	74










Platform Username Node	75
Profile Completeness Decision Node	75
Query Filter Decision Node	76
Required Attributes Present Node	76
Select Identity Provider Node	77
Social Provider Handler Node	78
Terms and Conditions Decision Node	79
Time Since Decision Node	79

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-7.1.0.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 "*Social Authentication*".

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 "*Social Authentication*".

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

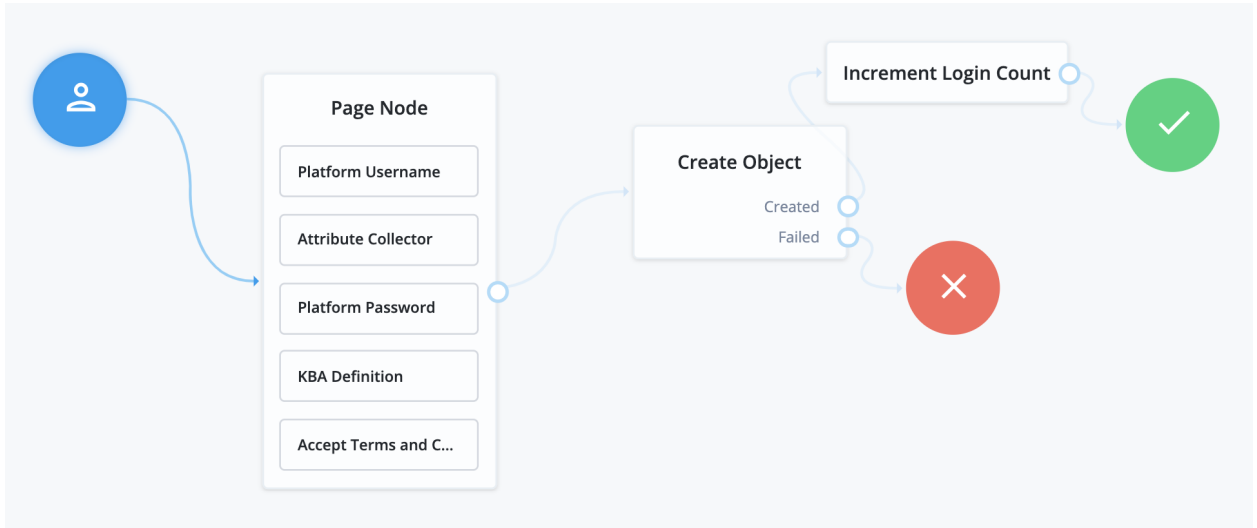
User Self-Registration lets your users create their own accounts. To configure registration, your registration tree requires at least the following nodes:

- 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; users must accept the terms before they are able to progress in the account creation process.

To set up Terms and Conditions in the Platform Admin UI:

1. Select Terms & Conditions and click + New Version.
2. Enter a version number for the new Terms and Conditions, then click Next.

Terms and Conditions are 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 dates.

3. Enter the locale for which these Terms and Conditions apply, expressed as its ISO 639-1 code (for example, `en` or `fr`), then click Add.
4. Enter the text of your Terms and Conditions:
 - Terms and Conditions are formatted using Markdown. Click Styles to apply additional CSS formatting to the HTML that is rendered from the Markdown.
 - The text supports localization. When you have added the Terms and Conditions for this locale, click Locale: `locale-name`, then click + Add locale to add the text for another locale.
 - Click Try it out to see how your Terms and Conditions will appear to your users.
5. Save or publish the new version.

Caution

When you have published a version, the Terms and Conditions cannot be edited. Be sure to proofread your text before publishing.

- Click Save as Draft to save this version for future publication. You can edit a draft version.
- Click Publish to publish this version.

Select Set as Active Version to make this the Active version of your Terms and Conditions. Only one version of Terms and Conditions can be active at a time, for each locale. Selecting this option will deactivate the currently active version, and make this version active instead.

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": "IDTokenIvalidateOnly",
    "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"
      ]
    }
  }
},
"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": "echoOn",
      "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

Social Authentication

You can configure user self-registration to include social identity providers as an option for users. This lets users register and log in to ForgeRock Identity Platform using an account they have through another trusted service.

The high-level steps to configure social authentication are:

- "Configure Social Identity Providers"
- "Configure a Basic Social Registration Tree"
- "Configure Social Registration with Account Claiming"

Configure Social Identity Providers

ForgeRock Identity Platform supports social identity providers that are OAuth 2.0 or OpenID Connect 1.0-compliant, and comes preconfigured with support for a number of social providers:

Default Social Identity Provider Configurations

Amazon	Apple	Facebook
Google	Instagram	itsme ^a
LinkedIn	Microsoft	Salesforce
Twitter	VK (Vkontakte)	WeChat
WordPress	Yahoo	-

^a To integrate with *itsme*, you must obtain an Organization Validation (OV) certificate and configure it in the container where AM runs, or in the reverse proxy offloading SSL.

If you need support for a social identity provider that is not preconfigured, you can manually add new providers, as long as they have a solution implemented using either OAuth 2.0, or OpenID Connect.

To Add Identity Providers

1. Register a service in the identity provider, and keep their documentation within reach. You will use it through this procedure.

Registering in a provider comprises creating a client ID and adding the redirection URL to ForgeRock Identity Platform at the very least.

+ About Redirection URLs

The redirection URL is a path in ForgeRock Identity Platform, usually in AM, that the identity provider will redirect the user to after a successful authentication. For example, <https://platform.example.com:8443/am>.

Depending on the social identity provider and on your environment, you may need to make changes to the redirection URI later.

Configure the same redirection URL in the identity provider service and in the ForgeRock Identity Platform client.

Some providers require that you enable a specific API in their service:

Google

Enable the [GMail API](#) in the Google Cloud Platform.

Apple

You must have access to the Apple Development Program (Enterprise program is not eligible), and you must enable [Sign In With Apple](#) in Apple Developer.

2. In the AM Admin UI, go to Realms > *Realm Name* > Services.
3. Check if the [Social Identity Provider Service](#) appears in the list of services configured for the realm.

If it does not, add it: Click on Add a Service, and select [Social Identity Provider Service](#) from the drop-down list.

The service's Configuration page appears.

4. Ensure that the Enabled switch is on.
5. Go to the Secondary Configurations tab.

ForgeRock Identity Platform includes scripts and configurations for several common identity providers.

6. In the Add a Secondary Configuration drop-down list, select the required identity provider.

If you do not see the required provider, select one of the following to add a custom identity provider client:

- Client Configuration for providers that implement the OAuth2 specification
- Client Configuration for providers that implement the OpenID Connect specification

The new identity provider configuration page appears.

7. Provide the client's required configuration details, such as the Client ID, Client Secret (for confidential clients), the Scope Delimiter (usually an empty space), and the Redirect URL

+ *About Redirection URLs*

The redirection URL is a path in ForgeRock Identity Platform, usually in AM, that the identity provider will redirect the user to after a successful authentication. For example, <https://platform.example.com:8443/am>.

Depending on the social identity provider and on your environment, you may need to make changes to the redirection URI later.

Configure the same redirection URL in the identity provider service and in the ForgeRock Identity Platform client.

Do not worry if you are missing some of the details; you will be able to edit the configuration later, after saving the client profile for the first time.

Save your changes to access all the configuration fields for the client.

8. Provide the client's advanced configuration details, and edit any required configuration details if needed.

+ *Where Do I Find the Required Identity Provider Information?*

- Refer to the provider's documentation.

Providers must specify their integration needs in their documentation, as well as their API endpoints.

For example, providers usually have different scopes that you can configure depending on your service's needs.

Financial-grade providers usually also require additional security-related configuration, such as `acr` values, PKCE-related settings, and more.

Keep their documentation close while configuring the client profile.

- Visit the provider's `.well-known` endpoint.

OAuth 2.0/OpenID Connect-compliant providers will display much of the information you need to configure the identity provider client in their `.well-known` endpoint. For example, the

endpoint should expose their endpoint URLs, and the signing and encryption algorithms they support.

ForgeRock Identity Platform is preconfigured for your convenience, but you must make sure the settings for the provider have not changed. Some of the most important preconfigured fields are:

- The provider's URLs. For example, Authentication Endpoint URL, Access Token Endpoint URL, and User Profile Service URL.
- The OAuth Scopes field.
- The configuration in the UI Config Properties section.
- The script selected in the Transform Script drop-down list.

Scripts named after identity providers are suitable for most use cases. However, if you need to view or edit the scripts, go to Realms > *Realm Name* > Scripts.

Note

Some features require choosing algorithms from those supported by the provider, as well as creating secrets. Consider the following points before configuring the client:

- Several capabilities in the identity provider client share the same secret IDs. For example, signing request objects and signing client authentication JWTs.
- Every identity provider client in a realm shares the same secrets.

Therefore, ensure that you configure features requiring secrets in a way that they are compatible across clients in the same realm.

For more information, see the page about the `/oauth2/connect/rp/jwk_uri` endpoint.

Expand the following link for tips on how to configure the client:

+ *Client Configuration Reference*

Enabled

Specifies whether the provider is enabled.

Required: Yes.

Auth ID Key

Specifies the attribute the social identity provider uses to identify an authenticated individual. For example, `id`, `sub`, and `user_id`.

Required: Yes.

Client ID

Specifies the `client_id` parameter as described in section 2.2 of *The OAuth 2.0 Authorization Framework* specification.

Required: Yes.

Client Secret

Specifies the `client_secret` parameter as described in section 2.3 of *The OAuth 2.0 Authorization Framework* specification.

Required: No.

Authentication Endpoint URL

Specifies the URL to the social provider's endpoint handling authentication as described in section 3.1 of *The OAuth 2.0 Authorization Framework*. For example, <https://accounts.google.com/oauth2/v2/auth>.

Required: Yes.

Access Token Endpoint URL

Specifies the URL to the endpoint handling access tokens as described in section 3.2 of *The OAuth 2.0 Authorization Framework* specification. For example, <https://www.googleapis.com/oauth2/v4/token>.

Required: Yes.

User Profile Service URL

Specifies the user profile URL that returns profile information. For example, <https://www.googleapis.com/oauth2/v3/userinfo>.

This URL should return JSON objects in its response.

Required: No.

Token Introspection Endpoint URL

Specifies the URL to the endpoint handling access token validation, as described in the OAuth 2.0 Token Introspection specification. For example, <https://oauth2.googleapis.com/tokeninfo>.

Required: No.

Redirect URL

Specifies the URL the identity provider will redirect the user to after authenticating, as described in Section 3.1.2 of *The OAuth 2.0 Authorization Framework* specification.

This URL is usually a page or path in AM; for example, `https://platform.example.com:8443/am`, and it is also registered in the identity provider's service.

You can also use a custom URI scheme as the redirect, if you are using an app built with the ForgeRock SDKs for Android or iOS. For example, `com.example.sdkapp:redirect_uri_path` or `frauth://com.forgerock.ios.sdkapp`.

Tip

When using the `FORM_POST` Response Mode, you must specify the `form_post` endpoint in the redirection URL. See [Response Mode](#) for more information.

Required: Yes.

Redirect after form post URL

Specifies the URL of a custom login page or application. ForgeRock Identity Platform will send processed form post data related to social login authentication to that URL as the value of the `form_post_entry` query parameter.

To continue the authentication journey, the custom login page is responsible for making a call to the ForgeRock Identity Platform `/json/authenticate` endpoint with the authentication ID (`authID`) and the processed form data (`form_post_entry`).

Configure this property when the following is true:

- The `FORM_POST` Response Mode is configured.
- Your users log in to ForgeRock Identity Platform using custom login pages, such as apps using the ForgeRock SDKs, instead of the ForgeRock Identity Platform UI.

Required: No.

Scope Delimiter

Specifies the delimiter used to separate scope values. For example, a blank space (), or a comma character (,).

Most providers use a blank space.

Required: Yes.

OAuth Scopes

Specifies the list of scopes to request from the provider.

The scopes that the provider returns depends on the permissions that the resource owner, such as the end user, grants to the client application.

For example, Google exposes its supported scopes in their OAuth 2.0 Scopes for Google APIs documentation.

Required: Yes.

Client Authentication Method

Specifies how the client should authenticate to the provider. Possible values are:

- **CLIENT_SECRET_POST**. The client sends the client ID and the secret in the `client_id` and the `client_secret` parameters in the body of the request.
- **CLIENT_SECRET_BASIC**. The client sends the client ID and the secret in a basic authorization header with the base64-encoded value of `client_id:client_secret`.
- **PRIVATE_KEY_JWT**. The client sends its credentials to the provider in a signed JWT as specified in the JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants.
- **ENCRYPTED_PRIVATE_KEY_JWT**. The client sends its credentials to the provider in a signed, then encrypted JWT as specified in the JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants.
- **TLS_CLIENT_AUTH**. The client presents a X.509 certificate that uses public key infrastructure (PKI), as specified in version 12 of the OAuth 2.0 Mutual TLS (mTLS) Client Authentication and Certificate Bound Access Tokens draft.
- **SELF_SIGNED_TLS_CLIENT_AUTH**. The client presents a X.509 self-signed certificate, as specified in version 12 of the OAuth 2.0 Mutual TLS (mTLS) Client Authentication and Certificate Bound Access Tokens draft.

Some of the authentication methods require additional configuration:

+ *How Do I Configure JWT Authentication With Signed JWTs?*

1. Obtain a list of supported signing algorithms from the provider's `.well-known` endpoint, and decide which one you will use.
2. In the Private Key JWT Signing Algorithm field, enter the signing algorithm that ForgeRock Identity Platform will use to sign the JWT. For example, `RSA256`.

This field may already be configured if the client is sending request objects.

3. Create a signing secret, and map it to the `am.services.oauth2.oidc.rp.jwt.authenticity.signing` secret ID in an AM secret store.

The secret ID may already have secrets mapped to it if the client is sending signed request objects to the provider, or if another client in the realm is already using it.

For more information, see [Configuring Secret Stores](#), and `/oauth2/connect/rp/jwk_uri`.

4. Provide a JWK with the public key to the identity provider. Refer to their documentation for more information.

For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

5. (Optional) Change the value in the Private Key JWT Expiration Time (seconds) field, if needed. It has a sensible value preconfigured, but you may need to tune it for your provider.

+ *How Do I Configure JWT Authentication With Signed and Encrypted JWTs?*

1. Follow the steps in [How Do I Configure JWT Authentication With Signed JWTs?](#) to configure ForgeRock Identity Platform to sign authentication JWTs.

Now you are ready to configure ForgeRock Identity Platform to encrypted authentication JWTs.

2. Obtain a list of supported encryption algorithms and methods from the provider's `well-known` endpoint, and decide which one you will use.
3. In the JWT Encryption Algorithm field, select the encryption algorithm.

If the required encryption algorithm does not appear in the drop-down, check the reference entry for the JWT Encryption Algorithm field for information on how to add it.

This field may already be configured if the client is encrypting request objects.

4. In the JWT Encryption Method field, select the encryption method.

This field may already be configured if the client is encrypting request objects.

5. In the JWKS URI Endpoint field, configure the URI containing the provider's public JWK set.

Obtain the URI from the provider's `.well-known` endpoint, or their documentation.

ForgeRock Identity Platform will use the JWK URI to fetch the provider's public encryption key.

6. (Optional) Perform one of the following steps depending on the encryption method you configured:
 - If you chose Direct AES Encryption method, select `NONE` in the JWT Signing Algorithm field. Signing is redundant with this encryption method.
 - If you chose an encryption method different from the Direct AES Encryption method, configure signing. For more information, see [How Do I Configure JWT Authentication With Signed JWTs?](#).

+ *How Do I Configure mTLS Authentication with PKI?*

1. Obtain a certificate for ForgeRock Identity Platform to use as a client. This certificate must be signed by a certificate authority (CA).

You can use the same certificate for different social identity provider client configurations, and you can only have one mTLS certificate by realm (either PKI-related, or self-signed).

2. Make the certificate available to ForgeRock Identity Platform configuring it in an AM secret store, and map its alias to the `am.services.oauth2.mtls.client.authentication` secret ID.

For example, you can create a `PKCS12` keystore secret store.

For more information, see [Configuring Secret Stores](#).

Even though the identity provider should trust the CA certificate automatically, the client certificate will appear in the `/oauth2/connect/rp/jwk_uri` endpoint.

+ *How Do I Configure mTLS Authentication with Self-Signed Certificates?*

1. Obtain a self-signed certificate that ForgeRock Identity Platform will use as a client.

You can use the same certificate for different social identity provider client configurations, and you can only have one mTLS certificate by realm (either PKI-related, or self-signed).

2. Make the certificate available to ForgeRock Identity Platform configuring it in an AM secret store, and map its alias to the `am.services.oauth2.mtls.client.authentication` secret ID.

For example, you can create a `PKCS12` keystore secret store.

For more information, see [Configuring Secret Stores](#).

To trust the self-signed certificate, the social identity provider must be able to access its public key and certificate. Social identity providers may have different ways of accessing public keys; for example, you may be able to configure the public JWK directly in the provider, or you may be able to provide ForgeRock Identity Platform's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes it.

Refer to your social identity provider documentation for more information.

Required: Yes.

PKCE Method

Specifies the PKCE transformation method ForgeRock Identity Platform uses when making requests to the provider's authorization endpoint, as specified in Section 4.2 of the *Proof Key for Code Exchange by OAuth Public Clients* specification.

Select `NONE` to disable PKCE transformations.

Required: No.

Request Parameter JWT Option

(OpenID Connect providers only) Specifies whether ForgeRock Identity Platform should provide a request object JWT to the provider. Possible values are:

- `NONE`. ForgeRock Identity Platform does not send a request object to the provider.
- `REFERENCE`. The request object JWT is stored in AM's CTS token store, and ForgeRock Identity Platform exposes a unique identifier for it using the `oauth2/request_uri` endpoint for the realm. The URL to the endpoint and the JWT's unique identifier are passed to the provider in the `request_uri` parameter of the request.

Ensure that the provider can reach the endpoint.

An example of the URL is https://platform.example.com:8443/am/realms/root/realms/myRealm/oauth2/request_uri/requestobjectID

Note

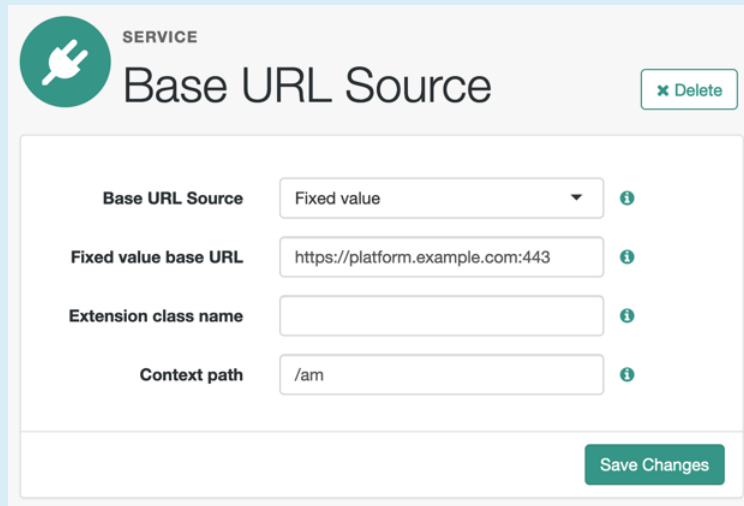
When integrating with *itsme*, ensure that the base URL of AM contains the 443 port. For example, <https://platform.example.com:443/am>.

To do this, configure the reverse proxy or load balancer to expose the port, or the Base URL Source Service:

+ *Base URL Source Example*

In the AM Admin UI, go to Realms > *Realm Name* > Services.

Add a Base URL Source service if one is not already configured, or select it to change its properties:



- **VALUE.** ForgeRock Identity Platform appends the JWT as the value of the `request` parameter of the request.

+ *How Do I Configure the Client to Send Signed Request Objects?*

1. In the Request Parameter JWT Option field, select either **VALUE** or **REFERENCE**.

Refer to your identity provider's documentation for more information.

2. Obtain a list of supported signing algorithms from the provider's `.well-known` endpoint, and decide which one you will use.
3. In the JWT Signing Algorithm field, select the signing algorithm that ForgeRock Identity Platform will use to sign the request object. For example, `RS256`.

This field may already be configured if the client is using JWT client authentication.

4. Create a signing secret that uses the algorithm you selected previously, and map it to the `am.services.oauth2.oidc.rp.jwt.authenticity.signing` secret ID in an AM secret store.

The secret ID may already have secrets mapped to it if the client is using JWT client authentication, or if another client in the realm is already using it.

For more information, see [Configuring Secret Stores](#), and `/oauth2/connect/rp/jwk_uri`.

5. Provide a JWK with the public key to the identity provider. Refer to the their documentation for more information.

For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

+ *How Do I Configure the Client to Send Signed and Encrypted Request Objects?*

1. Follow the steps in [How Do I Configure the Client to Send Signed Request Objects?](#) to configure ForgeRock Identity Platform to send signed request objects.

Now you are ready to configure ForgeRock Identity Platform to send encrypted request objects.

2. Enable Encrypt Request Parameter JWT.
3. Obtain a list of supported encryption algorithms and methods from the provider's `.well-known` endpoint, and decide which one you will use.
4. In the JWT Encryption Algorithm field, select the encryption algorithm.

If the required encryption algorithm does not appear in the drop-down, check the reference entry for the JWT Encryption Algorithm field for information on how to add it.

This field may already be configured if the client is encrypting authentication JWTs.

5. In the JWT Encryption Method field, select the encryption method.

This field may already be configured if the client is encrypting authentication JWTs.

6. In the JWKS URI Endpoint field, configure the URI containing the provider's public JWK set.

Obtain the URI from the provider's `.well-known` endpoint.

ForgeRock Identity Platform will use the JWK URI to fetch the provider's public encryption key.

7. (Optional) Perform one of the following steps depending on the encryption method you configured:
 - If you chose Direct AES Encryption method, select `NONE` in the JWT Signing Algorithm field. Signing is redundant with this encryption method.
 - If you chose an encryption method different from the Direct AES Encryption method, configure signing. For more information, see [How Do I Configure the Client to Send Signed Request Objects?](#).

Encrypt Request Parameter JWT

Specifies whether the request parameter must be encrypted when Request Parameter JWT Option is set to `REFERENCE` or `VALUE`.

ACR Values

(OpenID Connect providers only) Specifies a space-separated list, in order of preference, of the client's `acr` values.

Required: No.

Well Known Endpoint

(OpenID Connect providers only) Specifies the URL for retrieving information about the provider, such as endpoints, and public keys. For example, <https://accounts.google.com/.well-known/openid-configuration>.

Required: Yes.

Request Object Audience

(OpenID Connect providers only) Specifies the intended audience (`aud`) of the request object when the Request Parameter JWT Option field is set to `VALUE` or `REFERENCE`.

When not configured, the value of the Issuer field will be used as the audience of the request object.

OP Encrypts ID Tokens

(OpenID Connect providers only) Specifies whether the provider encrypts ID Tokens.

+ *How Do I Configure the ForgeRock Identity Platform to Receive Encrypted Tokens?*

1. Obtain a list of supported ID token encryption algorithms from the provider's `.well-known` endpoint, and decide which one the client will use.
2. Create a suitable secret for the algorithm that you chose, and map it to the `am.services.oauth2.oidc.rp.idtoken.encryption` secret ID in an AM secret store.

The secret ID may already have secrets mapped if another client in the realm is already using it.

For more information, see [Configuring Secret Stores](#), and `/oauth2/connect/rp/jwk_uri`.

3. Provide a JWK with the public key to the identity provider. Refer to their documentation for more information.

For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint, which exposes the client's public keys.

Configure the realm's `/oauth2/connect/rp/jwk_uri` endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

Required: No.

Issuer

(OpenID Connect providers only) Specifies the issuer of ID Tokens. Must exactly match the value returned in the ID token.

Obtain the `issuer` value from the provider's `.well-known` endpoint.

Required: Yes.

Enable Native Nonce

(OpenID Connect providers only) When enabled, the provider native SDK must include a `nonce` claim in the ID token. The value of the claim must be the value of the `nonce` claim sent in the Authentication Request.

Required: No.

User Info Response Format

(OpenID Connect providers only) Specifies the format in which the provider's `userinfo` endpoint returns data. Possible values are:

- `JSON`. The provider's `userinfo` endpoint returns a JSON.
- `SIGNED_JWT`. The provider's `userinfo` endpoint returns a signed JWT.
- `SIGNED_THEN_ENCRYPTED_JWT`. The provider's `userinfo` endpoint returns a signed, then encrypted JWT.

Some of the options require additional configuration:

+ *How Do I Configure the Client to Receive Signed userinfo JWTs?*

- In the JWKS URI Endpoint field, configure the URL containing the provider's public JWK set. Obtain it from the provider's `.well-known` endpoint, or their documentation.

ForgeRock Identity Platform will use this URL to fetch the provider's public signing key.

+ *How Do I Configure the Client to Receive Signed, then Encrypted userinfo JWTs?*

1. Follow the steps in *How Do I Configure the Client to Receive Signed userinfo JWTs?* to configure ForgeRock Identity Platform to receive signed JWTs.

Now you are ready to configure ForgeRock Identity Platform to receive encrypted JWTs.

2. Obtain a list of supported ID token encryption algorithms from the provider's `.well-known` endpoint, and decide which one the client will use.
3. Create a suitable secret for the algorithm that you chose, and map it to the `am.services.oauth2.oidc.rp.idtoken.encryption` secret ID in an AM secret store.

The secret ID may already have secrets mapped if another client in the realm is already using it, or if the provider encrypts ID tokens.

For more information, see [Configuring Secret Stores](#), and [/oauth2/connect/rp/jwk_uri](#).

4. Provide a JWK with the public key to the identity provider. Refer to the their documentation for more information.

For example, you could copy the contents of the public JWK in a field in the provider's service configuration, or you could configure the realm's [/oauth2/connect/rp/jwk_uri](#) endpoint, which exposes the client's public keys.

Configure the realm's [/oauth2/connect/rp/jwk_uri](#) endpoint in the provider, which exposes the client's public keys. Refer to the provider's documentation for more information.

JWKS URI Endpoint

Specifies the URI that contains the public keys of the identity provider. ForgeRock Identity Platform will use these keys to verify signatures, or to encrypt objects.

Configure this field when:

- Client Authentication Method is set to [ENCRYPTED_PRIVATE_KEY_JWT](#).
- Encrypt Request Parameter JWT is enabled.
- User Info Response Format is set to [SIGNED_JWT](#) or [SIGNED_THEN_ENCRYPTED_JWT](#).

Required: No.

JWT Signing Algorithm

Specifies the signing algorithm supported by the provider that ForgeRock Identity Platform use to sign the following:

- Client authentication JWTs when Client Authentication Method is set to [PRIVATE_KEY_JWT](#).
- (OpenID Connect providers only) Request JWTs when Request Parameter JWT Option is set to [VALUE](#) or [REFERENCE](#).

Obtain a list of the supported algorithms from the provider's [.well-known](#) endpoint.

Select [NONE](#) if the client will encrypt the JWT with the Direct AES Encryption method, because the signature will be redundant.

Required: No.

JWT Encryption Algorithm

Specifies the encryption algorithm supported by the provider that ForgeRock Identity Platform should use to encrypt the following:

- Client authentication JWTs when Client Authentication Method is set to `PRIVATE_KEY_JWT`.
- (OpenID Connect providers only) Request JWTs when Request Parameter JWT Option is set to `VALUE` or `REFERENCE`.

If set to `NONE`, ForgeRock Identity Platform will not encrypt the JWTs.

Obtain a list of the supported algorithms from the provider's `.well-known` endpoint.

Configure the algorithms exposed in this field using the AM advanced server property, `openam.private.key.jwt.encryption.algorithm.whitelist`.

+ *How Do I Configure Advanced Server Properties?*

- To configure advanced server properties in the AM Admin UI for all AM instances, go to Configure > Server Defaults > Advanced.
- To configure advanced server properties for a particular instance, go to Deployment > Servers > *Server Name* > Advanced.

If the property you want to add or edit is not already configured, add it with its value, then click on the plus (+) button.

If the property you want to add or edit is already configured, click on the pencil (✎) button to edit it. When you are finished, click on the tick (✓) button.

Save your changes.

Required: No.

JWT Encryption Method

Specifies the encryption algorithm supported by the provider that ForgeRock Identity Platform should use to encrypt the following:

- Client authentication JWTs when Client Authentication Method is set to `PRIVATE_KEY_JWT`.
- (OpenID Connect providers only) Request JWTs when Request Parameter JWT Option is set to `VALUE` or `REFERENCE`.

Use in conjunction with `JWT Encryption Algorithm`.

Obtain a list of the supported methods from the provider's `.well-known` endpoint.

Required: No.

Private Key JWT Expiration Time (seconds)

Specifies the amount of time, in seconds, that ForgeRock Identity Platform will cache the client authentication JWT before creating a new one.

Caching the JWT avoids creating a new one for every client authentication. However, it may also become invalid if the provider changes its configuration.

Required: No.

Response Mode

(OpenID Connect providers only) Specify the way the provider will return ID tokens to ForgeRock Identity Platform. Possible values are:

- **DEFAULT**. The provider returns the ID token as query parameters, as explained in the OpenID Connect Core 1.0 incorporating errata set 1 specification.

Most preconfigured providers use the **DEFAULT** response mode.

- **FORM_POST**. The provider returns the ID token by submitting an HTML form using the HTTP POST method, as explained in the OAuth 2.0 Form Post Response Mode specification.

When using this response mode, add the `/oauth2/client/form_post/ClientConfigName` URI to the Redirect URL, where *ClientConfigName* is the name of the social identity provider client that you are configuring. For example, `https://platform.example.com:8443/am/oauth2/client/form_post/myAppleClient`.

By default, the `form_post` endpoint processes the post data, encrypts it, and redirects with it back to the authentication tree to resume authentication.

However, environments using custom login pages need to configure the Redirect after form post URL property to redirect back to the custom login pages.

Important

The `/oauth2/client/form_post` does not require authentication. Protect it from denial of service (DoS) attacks by limiting the rate at which it can take connections in your load balancer or proxy.

Moreover, if you configured ForgeRock Identity Platform with AES Key Wrap encryption, ensure that you configure the `org.forgerock.openam.encryption.useextractandexpand` property.

For more information, see [Preparing AES Key Wrap Encryption](#).

Required: Yes.

UI Config Properties

Specifies a map of properties defined and consumed in the UI. The map affects how the identity provider's logo will show on the login page.

+ *ForgeRock Identity Platform Common End User UI Properties*

- **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`).

Required: Yes.

Transform Script

Specifies a script to convert the provider's raw profile object into a normalized object. An authentication tree will later convert the object again into attributes the ForgeRock Identity Platform can use.

ForgeRock Identity Platform provides scripts for the preconfigured identity providers; they are ready to use, and suitable for most use cases.

To write a script in Groovy or Javascript for an identity provider, go to Realms > *Realm Name* > Scripts, and use the provided scripts as a reference.

+ *Transformation Script Information and Example*

The following is the default Groovy transformation script for Google:

```
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)))
```

The script returns a JSON object with all the mapped objects.

Each field is a map with the following format: (*"platformAttributeName", rawProfile.providerAttributeName*).

For example, *id* is the platform attribute name, while *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.

Important

The social authentication nodes expect every attribute to have a value. In other words, the attributes returned by the identity provider cannot be empty, or *null*. If any of the attributes is empty or *null*, the social authentication tree journey will end with an error.

For example, if a user tries to log in using Google as the identity provider, but they did not configure a surname in their account, Google will return *null* as the value of the *familyName* for the identity, and social authentication will fail.

Ensure that all the users have their social profiles configured correctly, or modify the transformation scripts so that they not collect attributes that may be empty.

Required: Yes.

9. Save your changes.

You are ready now to "Configure a Basic Social Registration Tree".

Tip

To let AM contact Internet services through a proxy, see [Configuring AM for Outbound Communication](#).

You can control the behavior of the connection factory that AM uses as a client of the social identity providers:

+ *Client Connection Handler Properties*

The following advanced server properties control different aspects of the connection factory:

- `org.forgerock.openam.httpclienthandler.system.clients.connection.timeout`
- `org.forgerock.openam.httpclienthandler.system.clients.max.connections`
- `org.forgerock.openam.httpclienthandler.system.clients.pool.ttl`
- `org.forgerock.openam.httpclienthandler.system.clients.response.timeout`
- `org.forgerock.openam.httpclienthandler.system.clients.retry.failed.requests.enabled`
- `org.forgerock.openam.httpclienthandler.system.clients.reuse.connections.enabled`

They have sensible defaults configured, but if you need to change them, see [Advanced Properties](#).

Configure a Basic Social Registration Tree

There are two nodes associated with Identity Providers:

Select Identity Provider Node

The "Select Identity Provider Node" 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.

ForgeRock Identity Platform includes a transformation script called Normalized Profile to Managed User, which this node uses to transform the identity object gathered from the identity provider into a ForgeRock Identity Platform 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.

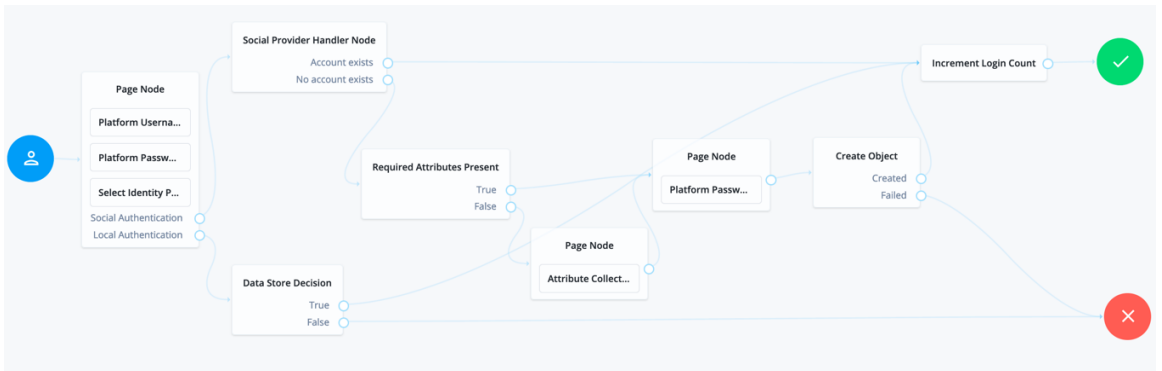
To Configure a Basic Social Registration Tree

1. In your realm, go to Journeys.

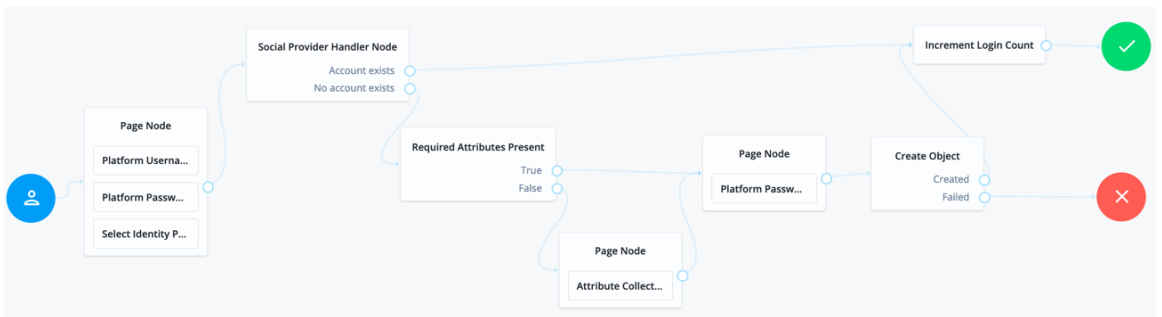
You can create a new tree, modify an existing tree, or duplicate an existing tree.

2. Decide whether users can log in with their local credentials, and add the relevant nodes to the tree:

- Social authentication trees allowing local authentication might look like the following:



- Social authentication trees enforcing social authentication login might look like the following:



To configure either option, use the Include local authentication switch in the "Select Identity Provider Node".

Note that, to support both local and social authentication in the same page, you must use the "Page Node" as shown in the example.

3. Configure the "Social Provider Handler Node":

- In the Transformation Script field, configure `Normalized Profile to Managed User`. This script will transform the normalized identity provider's profile object into an appropriate object that ForgeRock Identity Platform can use.

Find the script in Realms > *Realm Name* > Scripts.


- In Client Type, select `BROWSER` when using the ForgeRock Identity Platform UI, or the ForgeRock SDK for JavaScript.

4. Configure the "Required Attributes Present Node" and the "Create Object Node":

In the Identity Resource field, configure the relevant managed identity resource type. For example, `managed/user` or `managed/alpha_user`.

Tip

To check for the available managed identity resource types, go to the Identity Management Admin UI, and open the Manage drop-down list, at the upper right corner of the screen.

Identity managed object types are preceded by the  icon.

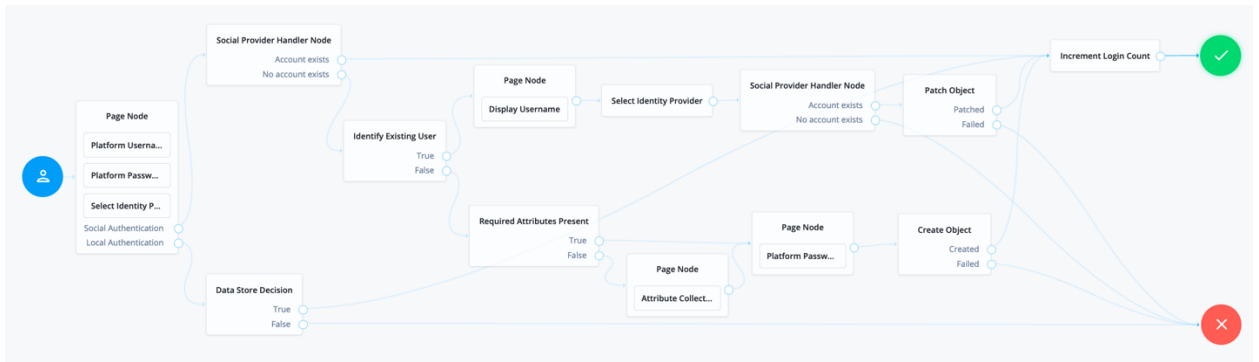
5. Configure the "Attribute Collector Node" adding, at least, the `mail`, `givenName`, and `sn` attributes.

Configure Social Registration with Account Claiming

If your users have one or more social identity provider accounts, they can link them to the same ForgeRock Identity Platform account. For more information, see [Account Claiming: Links Between Accounts and Social Identity Providers](#).

The following example builds on the basic social registration tree shown in "To Configure a Basic Social Registration Tree":

Example Social Registration Tree with Account Claiming



The tree uses the "Identify Existing User Node" to determine if the user is already registered in ForgeRock Identity Platform. By default, the node checks whether the email address associated with the account is already registered in ForgeRock Identity Platform.

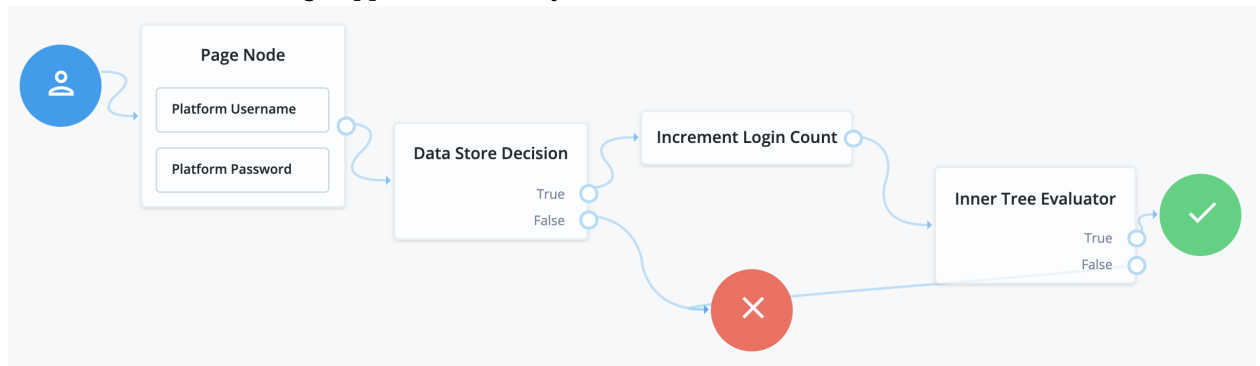
Ensure that you configure the Transformation Script in the "Social Provider Handler Node", and the Identity Resource field in the "Patch Object Node".

See "To Configure a Basic Social Registration Tree" for tips.

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 "[Social Authentication](#)".



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 "*Social Authentication*". 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.

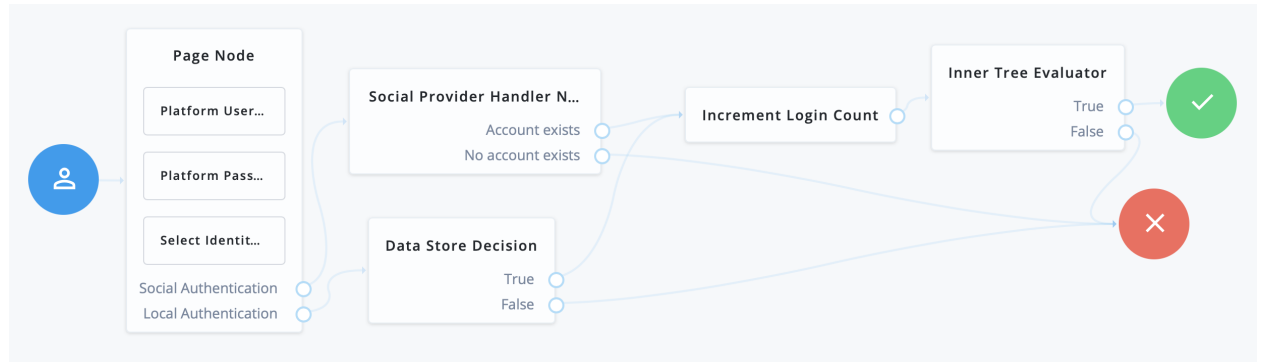
+ *This example will use the following nodes:*

- 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>"
}

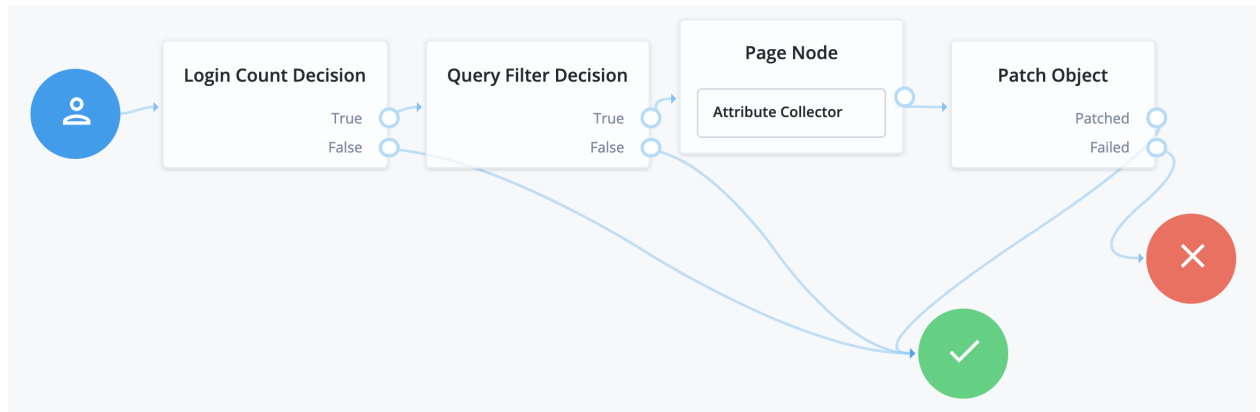
```

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

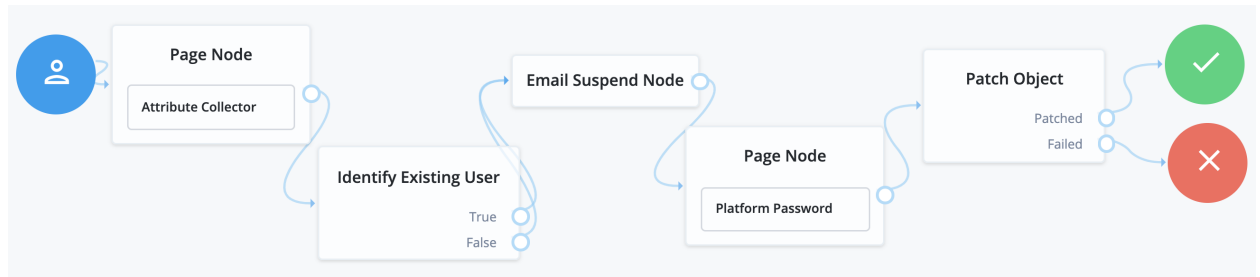
The Terms and Conditions Decision node verifies the user has accepted the currently active set of Terms and Conditions. Use this node when you want to verify the user has accepted your current terms and conditions before proceeding. Use this with the Accept Terms and Conditions node: connect the Terms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



Make sure that the Patch Object node's Patch As Object field is not selected (equivalent to `false`).

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>"
}
```

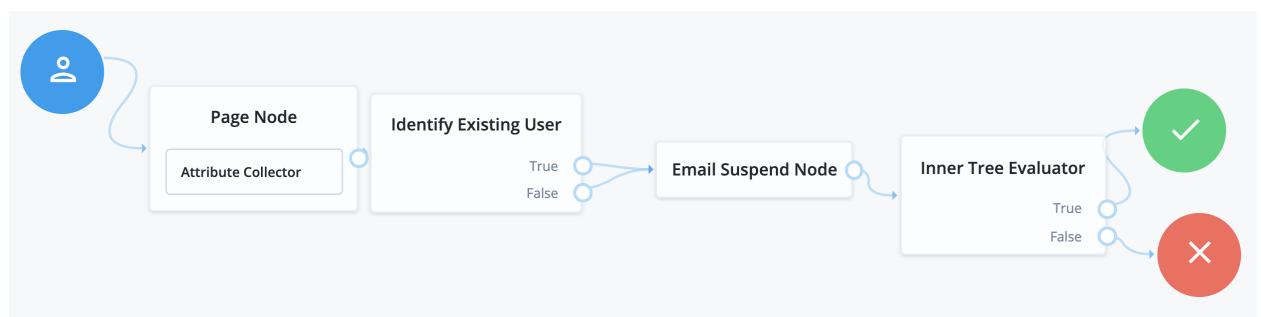
Chapter 8

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": [
```

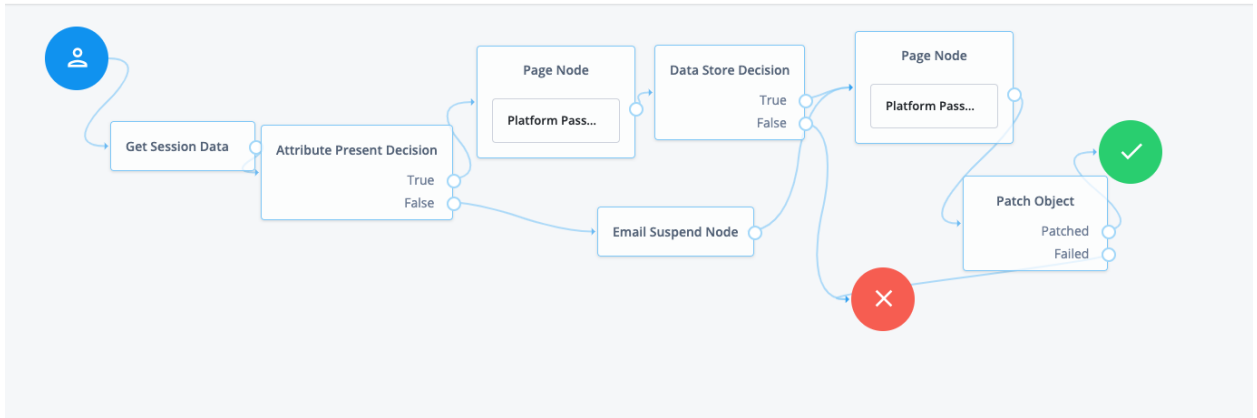


```
{
  {
    "name": "name",
    "value": "mail"
  },
  {
    "name": "prompt",
    "value": "Email Address"
  },
  {
    "name": "required",
    "value": true
  },
  {
    "name": "policies",
    "value": {}
  },
  {
    "name": "failedPolicies",
    "value": []
  },
  {
    "name": "validateOnly",
    "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.



Make sure that the Patch Object node's Patch As Object field is not selected (equivalent to `false`).

Chapter 10

Platform Authentication Nodes Reference

This page includes the configuration reference for authentication nodes that can only be used in a platform environment.

For information about all the nodes that AM supports, see the [Authentication Node Configuration Reference](#) in the *ForgeRock Access Management Authentication Guide*.

Accept Terms and Conditions Node

This node prompts the user to accept the currently active Terms and Conditions.

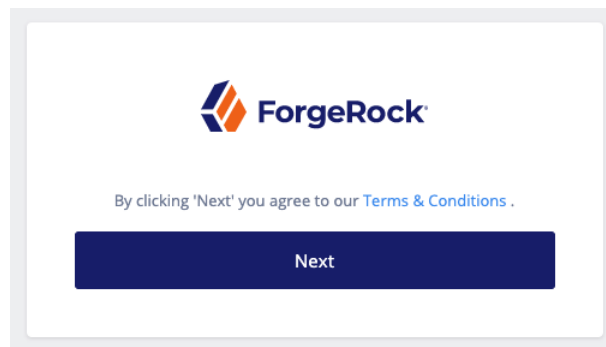
Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

You set up Terms and Conditions in the Platform UI. For more information, see [Configure Terms and Conditions](#) in the *Platform Self-Service Guide*.

This node is used in a registration tree, or combined with the "Terms and Conditions Decision Node" in a progressive profile or login tree.

Note that there is no failure path for this node: the user must accept the Terms and Conditions in order to proceed:



Properties:

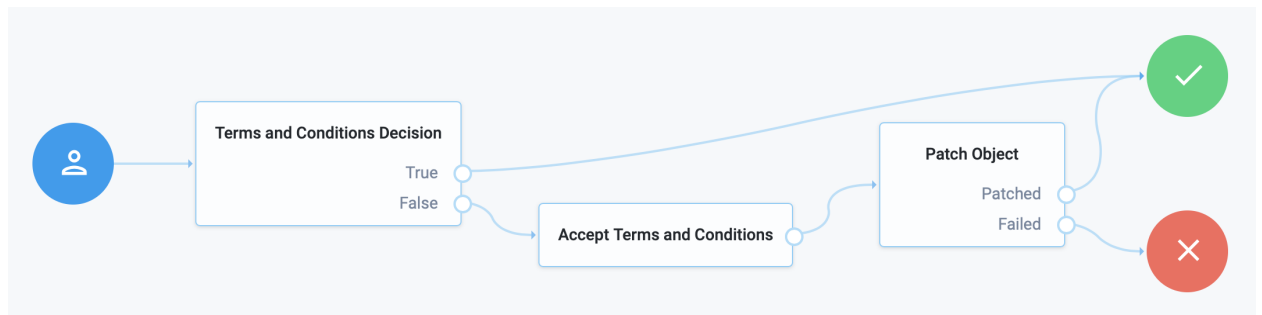
This node has no configurable properties.

Example:

In a progressive profile tree, the Accept Terms and Conditions node is used after the "Terms and Conditions Decision Node". If the user has not accepted the latest version of the Terms and Conditions, they are taken to a page notifying them that proceeding indicates accepting the current Terms and Conditions.

If the user clicks next, the acceptance response is stored in IDM.

Example Tree With Accept Terms and Conditions Node



Attribute Collector Node

The Attribute Collector node is used to collect the values of attributes for use elsewhere in a tree, such as collecting user information to populate a new account in a registration tree.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

To request a value, the attribute must be present in the IDM schema of the Identity Object configured in the tree. This node supports three types of attributes: **string**, **boolean**, and **number**.

The node configuration allows the admin to specify if the attributes are required to continue, and if they should be subject to validation through IDM's policy filter.

You can place the node anywhere in your authentication tree, or within a page node.

Properties:

Property	Usage
Attributes to Collect	A list of the attributes you wish to collect, based on the attributes found in the IDM schema for the identity object configured in the tree.

Property	Usage
All Attributes Required	When enabled, all attributes collected in this node are required in order to continue.
Validate Input	When enabled, the content input in this node should be validated against IDM policy settings specified in the IDM schema.
Identity Attribute	The attribute used to identify the object in IDM.

Attribute Present Decision Node

Checks if an attribute is present on an object, regardless of whether the field is private. Use this to verify an attribute is present, without needing to know the value of the attribute itself.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

A good use case is during an update password flow, where you want to check if the account has a password (rather than no password and logging in through a social identity) before continuing.

This node is similar to the "Attribute Value Decision Node" when that node is set to use the **PRESENT** operator, except it cannot return the value of the attribute, but can work with private attributes.

Properties:

Property	Usage
Present Attribute	The object attribute to verify is present in the IDM object. This can be an otherwise private attribute, such as password .
Identity Attribute	The attribute used to identify the object in IDM.

Attribute Value Decision Node

Verifies that the user's specified attribute satisfies a specific condition.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node to check whether an attribute's expected value is equal to a collected attribute value, or to validate that a specified attribute has been collected (regardless of the value of that attribute).

For example, to validate that a user filled out the country attribute when registering, set the comparison operation to **PRESENT**, and the comparison attribute to **country**.

If you instead need to ensure the country attribute is set to the United States, set the comparison operation to **EQUALS**, the comparison attribute to **country**, and the comparison value to **United States**.

Use "Attribute Present Decision Node" instead when you need to check for the presence of a private attribute (such as, **password**).

Properties:

Property	Usage
Comparison Operation	The operation to perform on the object attribute; PRESENT checks for existence of an attribute, EQUALS checks if the object's attribute value equals the configured comparison value.
Comparison Attribute	The object attribute to compare.
Comparison Value	This property is only relevant when using the EQUALS comparison operation, and is the value to compare the object's attribute value to.
Identity Attribute	The attribute used to identify the object in IDM.

Consent Collector Node

The Consent Collector node prompts the user to consent to share their profile data.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

A consent notice is listed for each IDM mapping that has consent enabled. If an IDM mapping is not created, or the mappings do not have privacy and consent enabled, no consent message will be shown to the user.

This node is primarily used in progressive profile or registration flows.

Properties:

Property	Usage
All Mappings Required	If enabled, all mappings listed by this node require consent in order to move forward.
Privacy & Consent Message	Localized message providing the privacy and consent notice. The key is the language (such as en or fr), and the value is the message to display.

Create Object Node


The Create Object node is used to create a new object in IDM based on information collected during an auth tree flow, such as user registration.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Any managed object attributes that are marked as required in IDM will need to be collected during the auth tree flow in order for the new object to be created.

Properties:

Property	Usage
Identity Resource	The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree. <div style="background-color: #e1f5fe; padding: 10px; margin-top: 10px;"> <p>Tip</p> <p>To check for the available managed identity resource types, go to the Identity Management Admin UI, and open the Manage drop-down list, at the upper right corner of the screen.</p> <p>Identity managed object types are preceded by the  icon.</p> </div>

Email Suspend Node

The Email Suspend node is used to generate and send an email to a user, such as an address verification email, based on an email template in IDM. The authentication tree will pause until the user clicks a link in the email to resume the tree flow.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The link is generated by the Email Suspend node, and is passed along to IDM as part of the email object, in a property called `resumeURI`.

This node uses the email service configured in IDM to send email. If you do not need the auth tree to pause and wait for a response from email, use the "Email Template Node" instead.

Properties:

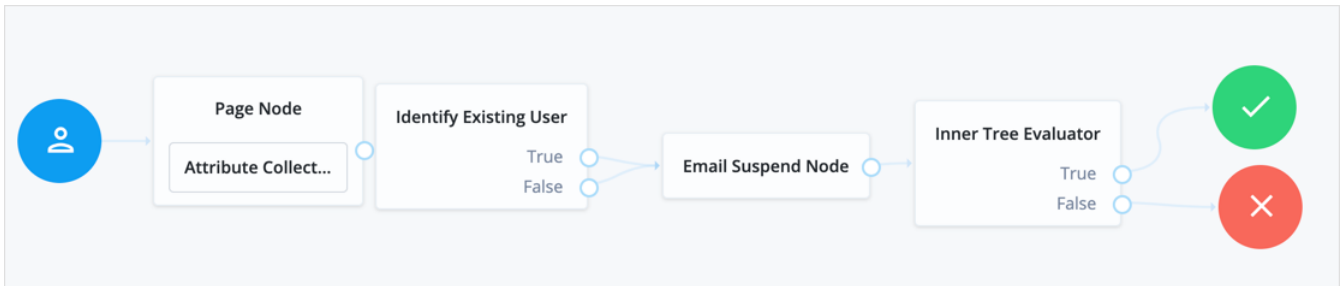
Property	Usage
Email Template Name	The name of the IDM email template to be sent. Check IDM for the names of available email templates, or to create a new template.
Email Attribute	The IDM attribute storing the address to send the email to.
Email Suspend Message	The localized message to be returned once the tree is suspended. The default message is "An email has been sent to your inbox."

Property	Usage
Object Lookup	Determines whether the object should be looked up in IDM. If true, IDM is queried for an existing object. Otherwise, the object in the authentication tree's shared state is used. For example, if suspending a user registration flow before the user object is created in IDM, this should be set to <code>false</code> . If the registration flow has already created the new user object when the flow is suspended, then this should be set to <code>true</code> .
Identity Attribute	The attribute used to identify the object in IDM.

Example:

The following is an example of a forgotten password tree. The user enters information that the Identify Existing User Node will use to try to identify them. Next, AM uses the "Email Suspend Node" to send an email to the user and suspend the authentication tree. Once authentication is resumed, the user is sent to a different tree to reset their password:

Email Suspend Tree



Email Template Node

The Email Template node is used to generate and send an email to a user, such as a welcome email, based on an email template in IDM.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This node uses the email service configured in IDM to send email. If you need the auth tree to pause and wait for a response from email, use the "Email Suspend Node" instead.

This node has two possible outcomes: "Email Sent" and "Email Not Sent", which can be used if you need different behavior depending on the outcome. According to OWASP authentication recommendations, the message to the user should be the same in both cases.

Properties:

Property	Usage
Email Template Name	The name of the IDM email template to be sent. Check IDM for the names of available email templates, or to create a new template.
Email Attribute	The IDM attribute storing the address to send the email to.
Identity Attribute	The attribute used to identify the object in IDM.

Identify Existing User Node

This node verifies a user exists based on an identifying attribute, such as an email address, then makes the value of a specified attribute available in a tree's shared state.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

For example, use this node in a "Forgot Username" flow to fetch a username to email to the user. If you want to display the username on screen, use the Display Username Node instead.

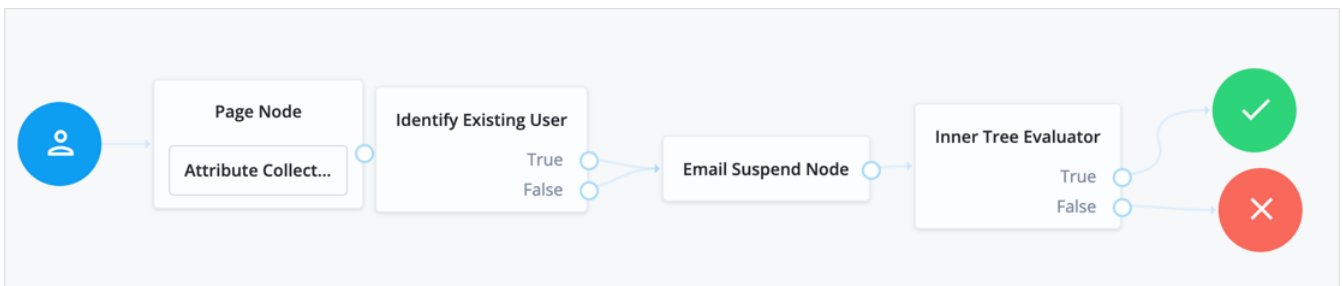
Properties:

Property	Usage
Identifier	The attribute to collect from an IDM object.
Identity Attribute	The attribute used to identify the object in IDM. Since this node is generally used for recovering a username, the identity attribute in this case should be some other attribute that is unique to a user object, such as the user's email address.

Example:

The following is an example of a forgotten password tree. The user enters information that the "Identify Existing User Node" will use to try to identify them. Next, AM sends the user an email, possibly with a link to resume authentication. Once authentication is resumed, the user is sent to a different tree to reset their password:

Identify User Tree



Increment Login Count Node

Increments the successful login count property of a managed object in IDM.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node in conjunction with the "Login Count Decision Node". If you plan to track the number of logins, include this node in your login authentication flow, but you can safely omit it if you are not planning to use that functionality.

Properties:

Property	Usage
Identity Attribute	The attribute used to identify the object in IDM.

KBA Decision Node

The KBA Decision node is used to check if the minimum number of KBA questions required by the system are defined for the user.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The number of KBA questions is determined by the `minimumAnswersToDefine` property in `selfservice.kba.json` in IDM. This node is mainly used for Progressive Profile completion.

Properties:

Property	Usage
Identity Attribute	The attribute used to identify the object in IDM.

KBA Definition Node

The KBA Definition node collects KBA questions and answers from the user and saves them to the user object.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used when creating or updating a user with Knowledge-Based Authentication enabled. For more information, see *Configure Security Questions* in the *Platform Self-Service Guide*.

Properties:

Property	Usage
Purpose Message	A localised message describing the purpose of the data requested from the user.

KBA Verification Node

The KBA Verification node presents KBA questions to the user, collects answers to those questions, and verifies the input against the user's stored answers.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used during self-service actions such as *Forgot Password* or *Forgot Username*, where additional authentication is needed. The number of KBA questions is determined by the `minimumAnswersToVerify` property in `selfservice.kba.json` in IDM.

Properties:

Property	Usage
KBA Attribute	The IDM object attribute in which KBA questions and answers are stored.
Identity Attribute	The attribute used to identify the object in IDM.

Login Count Decision Node

Triggers an action when a user's successful login count property reaches a specified number.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

The action can either be triggered once, by setting the `interval` property to happen **AT** the set amount of successful login attempts; or set to occur **EVERY** time the specified number of additional successful login attempts occur.

Use this node in conjunction with the "Increment Login Count Node". The Increment Login Count Node needs to be present in your login authentication flow for the Login Count Decision Node to have the data necessary to trigger a decision.

Properties:

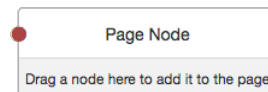
Property	Usage
Interval	The type of interval the decision should trigger on. Valid types are every and at . Every refers to a recurring action that happens every specified number of successful logins, such as prompting a user to update their contact information every 30 days. At refers to an action that occurs once, after the specified number of successful logins. For example, prompting the user to set their communication preferences once they have logged in 10 times.
Amount	The amount (count) of logins the interval should trigger on.
Identity Attribute	The attribute used to identify the object in IDM.

Page Node

The Page authentication node combines multiple nodes that request input into a single page for display to the user. Drag and drop nodes on to the page node to combine them.

The outcome paths are determined by the last node in the page node. Only the last node in the page can have more than one outcome path.

Only nodes that use callbacks to request input can be added to a Page Node. Other nodes, such as the Data Store Decision Node and Push Sender Node must not be added to a page node.



Properties:

Property	Usage
Header	Optional. Localized title for the page node and the nodes contained within it. Use this when components of an authentication flow need a title, such as breaking a registration into labeled sections.
Description	Optional. A localized description for the page node and the nodes contained within it. Use this when additional descriptive text is needed in an authentication flow.
Stage	Optional. This is used in UI development, to help identify what node or series of nodes are being returned so they can be rendered in the UI appropriately.

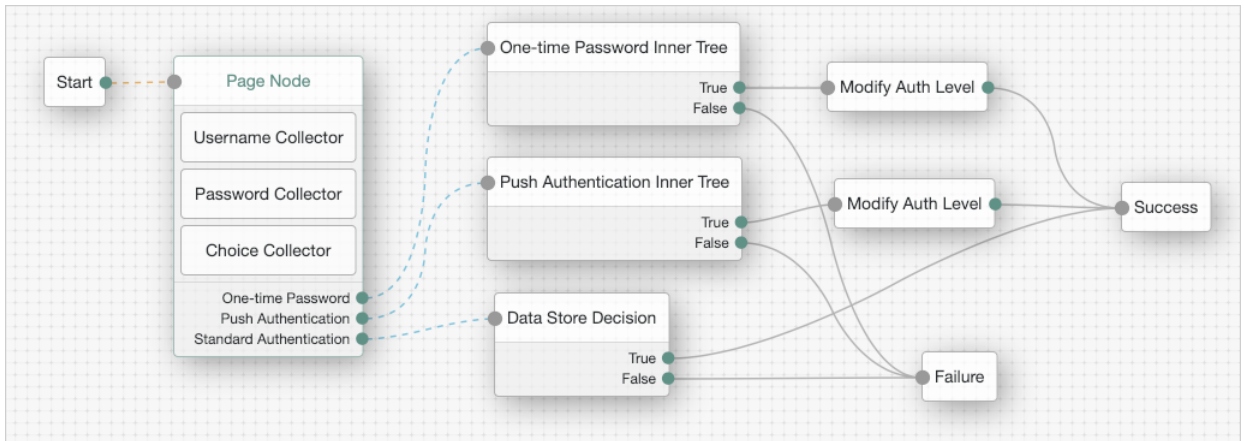
Note

The Page Node's optional properties are passed in the response, but the UI needs to support these properties before they will be visible to the end user.

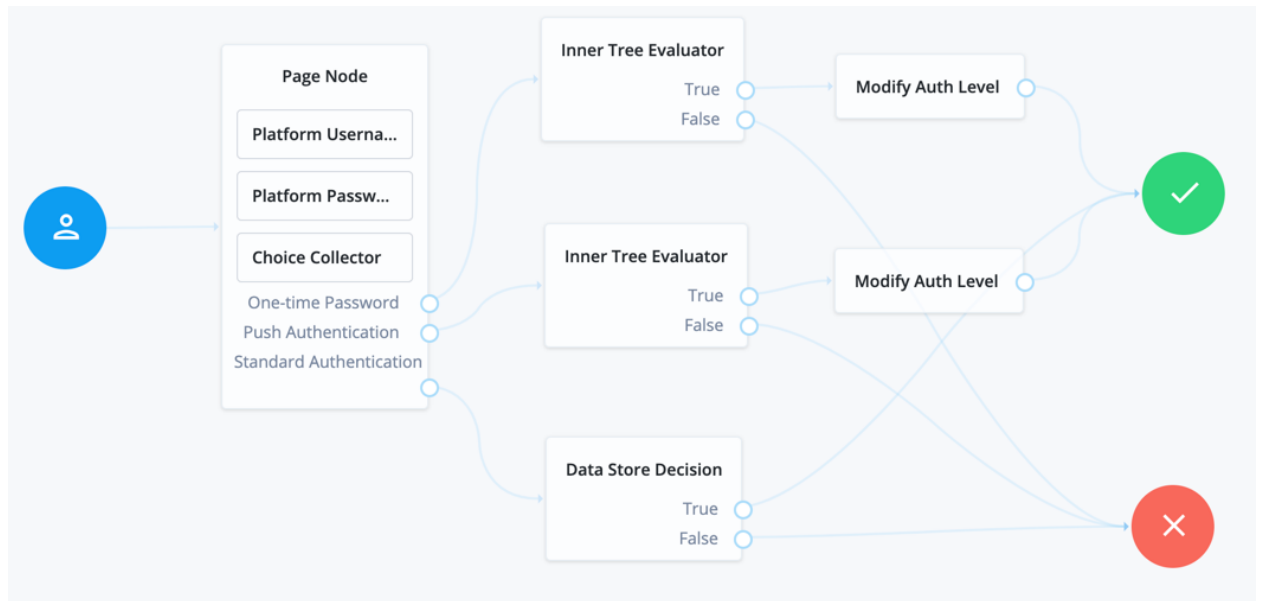
Example:

The following example uses a page node containing a username collector, a password collector, and a choice collector:

Example Tree With Page Node (Standalone AM)

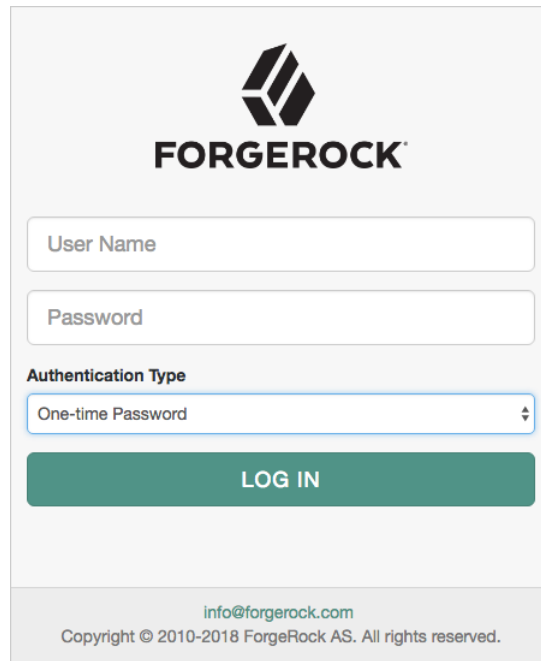


Example Tree With Page Node (ForgeRock Identity Platform)



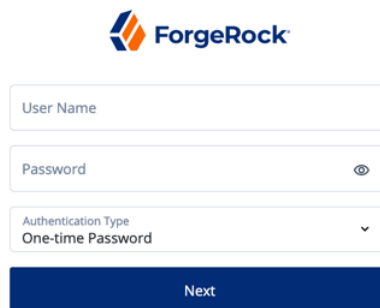
The user is presented with all of the requests for input on a single page:

User View of Example Tree with Page Node (Standalone AM)



The image shows a login form for a Standalone AM. At the top center is the ForgeRock logo. Below it are two input fields: "User Name" and "Password". Underneath the password field is a section titled "Authentication Type" with a dropdown menu currently set to "One-time Password". A large green button labeled "LOG IN" is positioned below the dropdown. At the bottom of the form, there is a footer containing the email address "info@forgerock.com" and the copyright notice "Copyright © 2010-2018 ForgeRock AS. All rights reserved."

User View of Example Tree with Page Node (ForgeRock Identity Platform)



The image shows a login form for the ForgeRock Identity Platform. At the top center is the ForgeRock logo. Below it are three input fields: "User Name", "Password" (with an eye icon for visibility), and "Authentication Type" (with a dropdown menu set to "One-time Password"). A large dark blue button labeled "Next" is positioned below the authentication type dropdown.

Patch Object Node


The Patch Object node is used to update attributes in an existing managed object in IDM.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

This is used in cases such as progressive profile completion, where you may wish to collect additional profile data from a user after they have logged in several times.

Properties:

Property	Usage
Patch as Object	Allows patching as the object being updated. Enable this property to patch a user object as part of the user's current session, such as when updating their password.
Ignored Fields	Fields from the tree's shared state that should be ignored as part of patch. If this is empty, all shared-state fields in tree's <code>nodeState</code> object are attempted as part of the patch. Use this to keep your patch focused only on the fields you want to update.
Identity Resource	<p>The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree.</p> <div style="background-color: #e1f5fe; padding: 10px; margin-top: 10px;"> <p>Tip</p> <p>To check for the available managed identity resource types, go to the Identity Management Admin UI, and open the Manage drop-down list, at the upper right corner of the screen.</p> <p>Identity managed object types are preceded by the  icon.</p> </div>
Identity Attribute	The attribute used to identify the object to update in IDM.

Platform Password Node

This node prompts the user to enter their password and stores the input in a configurable state attribute.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node instead of the Password Collector node when working with AM and IDM as an integrated platform.

Properties:

Property	Usage
Validate Password	<p>When enabled, this node checks the user's input against IDM's password policies, and returns any policy failures as errors. For example, if you submitted an invalid password on registration, the response from this node would include a list of failed policies:</p> <pre> { "name": "failedPolicies", "value": [{ "params": { "minLength": 8 }, "policyRequirement": "MIN_LENGTH" }, { "params": { "numCaps": 1 }, "policyRequirement": "AT_LEAST_X_CAPITAL_LETTERS" }, { "params": { "numNums": 1 }, "policyRequirement": "AT_LEAST_X_NUMBERS" }] }, </pre>
Password Attribute	The attribute used to store a password in the IDM object.

Platform Username Node

This node prompts the user to enter their username, and stores it in a configurable state attribute.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node instead of the Username Collector node when working with AM and IDM as an integrated platform.

Properties:

Property	Usage
Validate Username	When enabled, this node checks the user's input against IDM's username policies, and returns any policy failures as errors.
Username Attribute	The attribute used to store a username in the IDM object.

Profile Completeness Decision Node

The Profile Completeness Decision node is used in progressive profile flows. It checks how much of a user's profile has been filled out, where the completeness of a profile is expressed as a percentage of user-viewable, user-editable fields that are not `null`.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Properties:

Property	Usage
Profile Completeness Threshold	Percentage of user-viewable and user-editable fields in a profile that need to be filled out for the node to pass. Expressed as a number between 0 and 100.
Identity Attribute	The attribute used to identify the object in IDM.

Query Filter Decision Node

Checks if the contents of a user's profile matches a specified query filter.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Use this node to verify whether a particular field has been filled out, or that the contents of a field match a specific pattern. For instance, 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*.

Properties:

Property	Usage
Query Filter	A query filter used to check the contents of an object.
Identity Attribute	The attribute used to identify the object that will be queried in IDM.


Required Attributes Present Node

The Required Attributes Present node checks the specified identity resource in IDM (by default, `managed/user`), and determines if all attributes required to create the specified object exist within shared state of the tree.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

Properties:

Property	Usage
Identity Resource	<p>The type of IDM managed identity resource object that this node will create. It must match the identity resource type for the current tree.</p> <div style="background-color: #e0f2f1; padding: 10px; border: 1px solid #ccc;"> <p>Tip</p> <p>To check for the available managed identity resource types, go to the Identity Management Admin UI, and open the Manage drop-down list, at the upper right corner of the screen.</p> <p>Identity managed object types are preceded by the  icon.</p> </div>

Select Identity Provider Node

This node is used in combination with the "Social Provider Handler Node" to enable use of the Social Identity Provider Service. It presents the user with a list of configured, enabled, social identity providers to use for authentication.

It can also be configured to only show identity providers the user has already associated with their account, such as in account claiming flows, where a user wishes to associate a new social identity provider with an account that is being authenticated with social authentication.

The node has two possible outputs: social authentication, and local authentication. Local authentication can be turned off by disabling Include local authentication. In cases such as during account claiming, where the user has already authenticated once and is associating a new identity provider, the node will only display a local sign in option if it detects that the user's account has a `password` attribute present.

This node returns the `SelectIdPCallback` when more than one social identity provider is enabled, or a single provider is enabled as well as the Local Authentication option, and therefore a choice from the user is required. If no choice from the user is required, authentication proceeds to the next node in the tree.

Properties:

Property	Usage
Include local authentication	Determines whether local authentication will be included as an available method for authenticating.
Offer only existing providers	Enable this when the social identity provider choices offered should be limited to those already associated with a user object. Use this when a user is authenticating using a new social identity provider, and an account associated with that user already exists (also known as "account claiming").
Password attribute	The attribute in the user object that stores a user's password, for use during local authentication.

Property	Usage
Identity Attribute	The attribute used to identify an existing user. Required to support the offer of only existing providers.
Filter Enabled Providers	<p>By default, the node displays all identity providers that are marked as Enabled in the Social Identity Provider Service as a selectable option. Specify the name of one of more providers to filter the list.</p> <div style="background-color: #e0f2e0; padding: 10px; margin: 10px 0;"> <p>Tip</p> <p>View the names of your configured social identity providers by navigating to Services > Social Identity Provider Service > Secondary Configurations.</p> </div> <p>If this field is not empty, providers must be in the list, and also be enabled in the Social Identity Provider service, in order to be displayed. If left blank, all enabled providers are displayed.</p>

Social Provider Handler Node

This node is used alongside the "Select Identity Provider Node" to enable use of the Social Identity Provider Service.

It takes the provider selection from the "Select Identity Provider Node" and attempts to authenticate the user with that provider. It then collects relevant profile information from the provider and returns the user to the flow, and transforms that profile information into attributes ForgeRock Identity Platform can use.

Properties:

Property	Usage
Transformation Script	<p>A script that transforms a normalized social profile to an identity or managed object.</p> <p>Select Normalized Profile to Managed User, or any other script you have created for this purpose.</p>
Username Attribute	The attribute in IDM that contains the username for this object.
Client Type	<p>Specify the client type you are using to authenticate to the provider.</p> <p>Use the default, BROWSER, when making use of the ForgeRock-provided user interfaces, or the ForgeRock SDK for JavaScript. This causes the node to return the RedirectCallback.</p> <p>Select NATIVE if you are using the ForgeRock SDKs for Android or iOS. This causes the node to return the IdPCallback.</p>

Terms and Conditions Decision Node

The Terms and Conditions Decision node verifies the user has accepted the active set of Terms and Conditions.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

You set up Terms and Conditions in the Platform UI. For more information, see [Configure Terms and Conditions in the *Platform Self-Service Guide*](#).

Use this node when you want to verify the user has accepted your Terms and Conditions before proceeding (such as logging in, or in a progressive profile tree). This is often used with the "Accept Terms and Conditions Node".

Properties:

Property	Usage
Identity Attribute	The attribute used to identify the object to check in IDM.

Time Since Decision Node

Checks if a specified amount of time has passed since the user was registered.

Note

This functionality requires that you configure AM as part of a ForgeRock Identity Platform deployment.

For example, if you wanted to prompt users to review your terms and conditions after the account is a week old, you could set the **Elapsed Time** property to **10080** minutes. After that time has elapsed, the next time the user logs in, they will be prompted to review your terms and conditions.

This node is mainly used for Progressive Profile completion.

Properties:

Property	Usage
Elapsed Time	The amount of time since the user was created, in minutes, that needs to elapse before this node is triggered. This property also supports specifying basic time units. For example, when setting the property to 10080 minutes, writing 7 days or 1 week also works.
Identity Attribute	The attribute used to identify the object to update in IDM.