



Getting Started with Scripting

/ ForgeRock Access Management 7.0.1

Latest update: 7.0.1

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

Copyright © 2019-2021 ForgeRock AS.

Abstract

Introduction to the scripting engine provided by Access Management. ForgeRock Access Management provides intelligent authentication, authorization, federation, and single sign-on functionality.



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

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

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

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

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

DejaVu Fonts

Bitstream Vera Fonts Copyright

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

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

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

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

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

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

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

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

Arev Fonts Copyright

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

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

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

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

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

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

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

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

FontAwesome Copyright

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

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

Table of Contents

Overview	iv
1. The Scripting Environment	1
Security	4
Thread Pools	5
2. Scripting API Functionality	6
Accessing HTTP Services	6
Debug Logging	8
3. Managing Scripts (Console)	9
4. Managing Scripts (REST)	12
Querying Scripts	14
Reading a Script	16
Validating a Script	16
Creating a Script	17
Updating a Script	18
Deleting a Script	19
5. Managing Scripts (ssoadm)	20
Glossary	22

Overview

AM provides a scripting engine for you to develop scripts for client-side and server-side authentication, as well as for policy conditions, handling OpenID Connect claims, and others.

Quick Start



Getting Started with AM's Scripting Environment

Learn about how to use scripts in AM to modify the default AM behavior in specific situations.



Scripting Functionality

Discover which behaviors of AM you can modify with scripts.

About ForgeRock Identity Platform™ Software

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>.

Chapter 1

The Scripting Environment

AM supports scripts written in either JavaScript, or Groovy ¹, and the same variables and bindings are delivered to scripts of either language.

+ *How to determine the JavaScript Engine Version?*

You can use a script to check the version of the JavaScript engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

```
var rhino = JavaImporter(
    org.mozilla.javascript.Context
)

var currentContext = rhino.Context.getCurrentContext()
var rhinoVersion = currentContext.getImplementationVersion()

logger.error("JS Script Engine: " + rhinoVersion)

outcome = "true"
```

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `org.mozilla.javascript.Context`
- `org.forgerock.openam.scripting.timeouts.*`

To view the Java class whitelist, go to [Configure > Global Services > Scripting > Secondary Configurations](#). Select the script type, and on the Secondary Configurations tab, click `engineConfiguration`.

For information on the capabilities of the JavaScript engine AM uses, see [Mozilla Rhino](#).

+ *How to determine the Groovy Engine Version?*

You can use a script to check the version of the Groovy scripting engine AM is using. You could temporarily add the following script to a Scripted Decision node, for example, to output the engine version to the debug log:

¹Scripts used for client-side authentication must be written in JavaScript.

```
logger.error("Groovy Script Engine: " + GroovySystem.version)
outcome = "true"
```

Note

Ensure the following are listed in the Java class whitelist property of the scripting engine.

- `groovy.lang.GroovySystem`

To view the Java class whitelist, go to [Configure > Global Services > Scripting > Secondary Configurations](#). Select the script type, and on the [Secondary Configurations](#) tab, click `engineConfiguration`.

For information on the capabilities of the Groovy engine AM uses, see [Apache Groovy](#).

To access the functionality AM provides, import the required Java class or package, as follows:

JavaScript

```
var fr = JavaImporter(
    org.forgerock.openam.auth.node.api,
    javax.security.auth.callback.NameCallback
);
with (fr) {
    ...
}
```

Groovy

```
import org.forgerock.openam.auth.node.api.*;
import javax.security.auth.callback.NameCallback;
```

You may need to whitelist the classes you use in scripts. See "Security".

You can use scripts to modify default AM behavior in the following situations, also known as *contexts*:

Client-side Authentication

Scripts that are executed on the client during authentication. Client-side scripts must be in JavaScript.

Server-side Authentication

Scripts are included in an authentication module within a chain and are executed on the server during authentication.

Authentication Trees

Scripts are included in an authentication node within a tree and are executed on the server during authentication.

Policy Condition

Scripts used as conditions within policies.

OIDC Claims

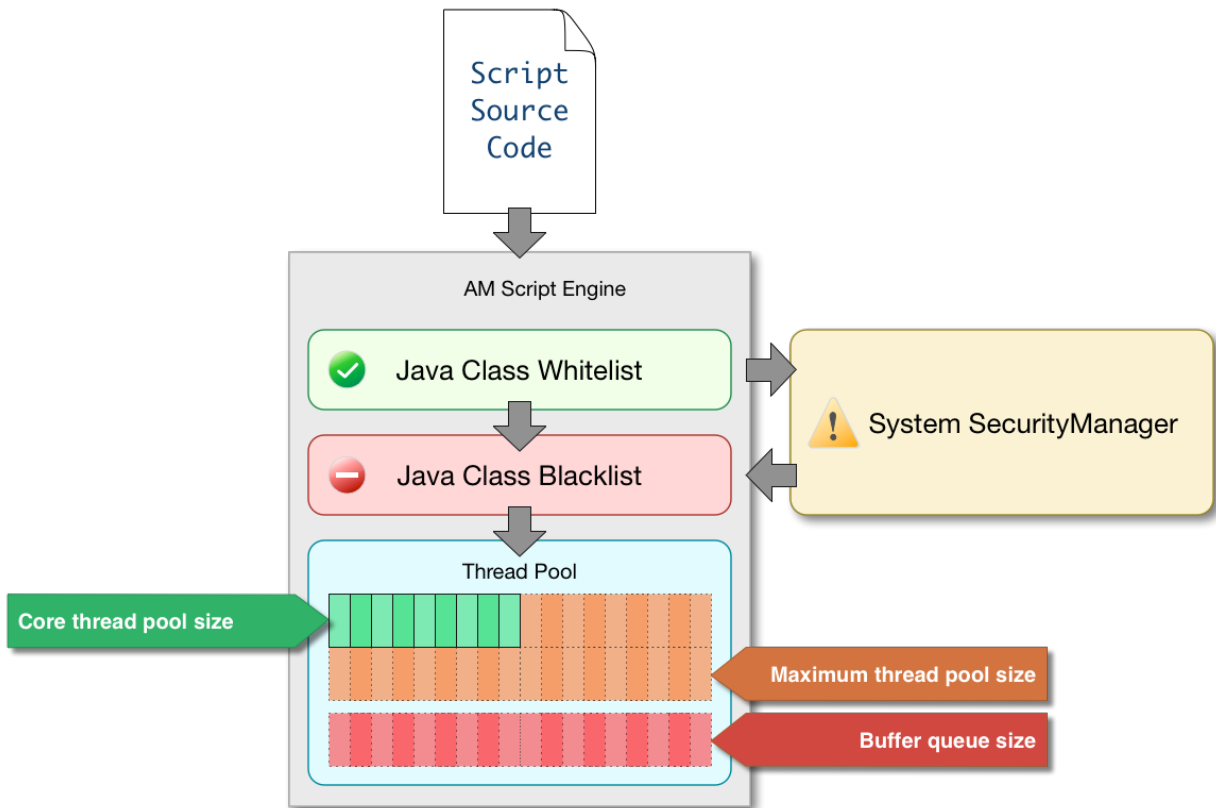
Scripts that gather and populate the claims in a request when issuing an ID token or making a request to the `userinfo` endpoint.

OAuth 2.0 Access Tokens

Scripts that modify the key-value pairs contained within access tokens before they are issued to a client.

AM implements a configurable scripting engine for each of the context types that are executed on the server.

The scripting engines in AM have two main components: security settings, and the thread pool.



Learn more:

- "Security"
- "Thread Pools"

Security

AM scripting engines provide security features for ensuring that malicious Java classes are not directly called. The engines validate scripts by checking all directly-called Java classes against a configurable blacklist and whitelist, and, optionally, against the JVM SecurityManager, if it is configured.

Whitelists and blacklists contain class names that are allowed or denied execution respectively. Specify classes in whitelists and blacklists by name or by using regular expressions.

Classes called by the script are checked against the whitelist first, and must match at least one pattern in the list. The blacklist is applied after the whitelist, and classes matching any pattern are disallowed.

You can also configure the scripting engine to make an additional call to the JVM security manager for each class that is accessed. The security manager throws an exception if a class being called is not allowed to execute.

For more information on configuring script engine security, see "Scripting" in the *Reference*.

Important Points About Script Engine Security

The following points should be considered when configuring the security settings within each script engine:

The scripting engine only validates directly accessible classes.

The security settings only apply to classes that the script *directly* accesses. If the script calls `Foo.a()` and then that method calls `Bar.b()`, the scripting engine will be unable to prevent it. You must consider the whole chain of accessible classes.

Note

Access includes actions such as:

- Importing or loading a class.
- Accessing any instance of that class. For example, passed as a parameter to the script.
- Calling a static method on that class.
- Calling a method on an instance of that class.
- Accessing a method or field that returns an instance of that class.

Potentially dangerous Java classes are blacklisted by default.

All Java reflection classes (`java.lang.Class`, `java.lang.reflect.*`) are blacklisted by default to avoid bypassing the security settings.

The `java.security.AccessController` class is also blacklisted by default to prevent access to the `doPrivileged()` methods.

Caution

You should not remove potentially dangerous Java classes from the blacklist.

The whitelists and blacklists match class or package names only.

The whitelist and blacklist patterns apply only to the exact class or package names involved. The script engine does not know anything about inheritance, so it is best to whitelist known, specific classes.

Thread Pools

Each script is executed in an individual thread. Each scripting engine starts with an initial number of threads available for executing scripts. If no threads are available for execution, AM creates a new thread to execute the script, until the configured maximum number of threads is reached.

If the maximum number of threads is reached, pending script executions are queued in a number of buffer threads, until a thread becomes available for execution. If a created thread has completed script execution and has remained idle for a configured amount of time, AM terminates the thread, shrinking the pool.

For more information on configuring script engine thread pools, see "[Scripting](#)" in the *Reference*.

Chapter 2

Scripting API Functionality

AM provides the following scripting functionality:

"Scripted Module API Functionality" in the *Authentication and Single Sign-On Guide*

(Authentication chains only) How to access authentication state data, user profile data, request data, and information gathered by client-side scripts.

"Scripted Decision Node API Functionality" in the *Authentication and Single Sign-On Guide*

(Authentication trees only) How to access data in request headers, shared state, and user session data.

"Policy Condition Script API Functionality" in the *Authorization Guide*

How to access the authorization state data, the information pertaining a session, and the user's profile data in authorization policies.

Scripting OpenID Connect 1.0 Claims in the *OpenID Connect 1.0 Guide*

How to map scopes to claims and data for OpenID Connect ID tokens.

"Modifying the Content of Access Tokens" in the *OAuth 2.0 Guide*

How to modify the key-value pairs contained within an OAuth 2.0 access token.

"Accessing HTTP Services"

How to configure the parameters for the HTTP client object in any server-side script.

"Debug Logging"

How to add debug logging to any server-side script.

Accessing HTTP Services

AM passes an HTTP client object, `httpClient`, to server-side scripts. Server-side scripts can call HTTP services with the `httpClient.send` method. The method returns an `HttpClientResponse` object.

Configure the parameters for the HTTP client object by using the `org.forgerock.http.protocol` package. This package contains the `Request` class, which has methods for setting the URI and type of request.

The following example, taken from the default server-side Scripted authentication module script, uses these methods to call an online API to determine the longitude and latitude of a user based on their postal address:

```
function getLongitudeLatitudeFromUserPostalAddress() {
    var request = new org.forgerock.http.protocol.Request();

    request.setUri("http://maps.googleapis.com/maps/api/geocode/json?address=" +
    encodeURIComponent(userPostalAddress));
    request.setMethod("GET");

    var response = httpClient.send(request).get();
    logResponse(response);

    var geocode = JSON.parse(response.getEntity());
    var i;

    for (i = 0; i < geocode.results.length; i++) {
        var result = geocode.results[i];
        latitude = result.geometry.location.lat;
        longitude = result.geometry.location.lng;

        logger.message("latitude:" + latitude + " longitude:" + longitude);
    }
}
```

HTTP client requests are synchronous and blocking until they return. You can, however, set a global timeout for server-side scripts. For details, see "Scripted Authentication Module Properties" in the *Authentication and Single Sign-On Guide*.

Server-side scripts can access response data by using the methods listed in the table below.

HTTP Client Response Methods

Method	Parameters	Return Type	Description
<code>HttpClientResponse.getCookies</code>	Void	<code>Map<String, String></code>	Get the cookies for the returned response, if any exist.
<code>HttpClientResponse.getEntity</code>	Void	<code>String</code>	Get the entity of the returned response.
<code>HttpClientResponse.getHeaders</code>	Void	<code>Map<String, String></code>	Get the headers for the returned response, if any exist.
<code>HttpClientResponse.getReasonPhrase</code>	Void	<code>String</code>	Get the reason phrase of the returned response.

Method	Parameters	Return Type	Description
<code>HttpClientResponse.getStatusCode</code>	Void	Integer	Get the status code of the returned response.
<code>HttpClientResponse.hasCookies</code>	Void	Boolean	Indicate whether the returned response had any cookies.
<code>HttpClientResponse.hasHeaders</code>	Void	Boolean	Indicate whether the returned response had any headers.

Debug Logging

Server-side scripts can write messages to AM debug logs by using the `logger` object.

AM does not log debug messages from scripts by default. You can configure AM to log such messages by setting the debug log level for the `amScript` service. For details, see "*Debug Logging*" in the *Maintenance Guide*.

The following table lists the `logger` methods.

Logger Methods

Method	Parameters	Return Type	Description
<code>logger.error</code>	<i>Error Message</i> (type: String)	Void	Write <i>Error Message</i> to AM debug logs if ERROR level logging is enabled.
<code>logger.errorEnabled</code>	Void	Boolean	Return <code>true</code> when ERROR level debug messages are enabled.
<code>logger.message</code>	<i>Message</i> (type: String)	Void	Write <i>Message</i> to AM debug logs if MESSAGE level logging is enabled.
<code>logger.messageEnabled</code>	Void	Boolean	Return <code>true</code> when MESSAGE level debug messages are enabled.
<code>logger.warning</code>	<i>Warning Message</i> (type: String)	Void	Write <i>Warning Message</i> to AM debug logs if WARNING level logging is enabled.
<code>logger.warningEnabled</code>	Void	Boolean	Return <code>true</code> when WARNING level debug messages are enabled.

Chapter 3

Managing Scripts (Console)

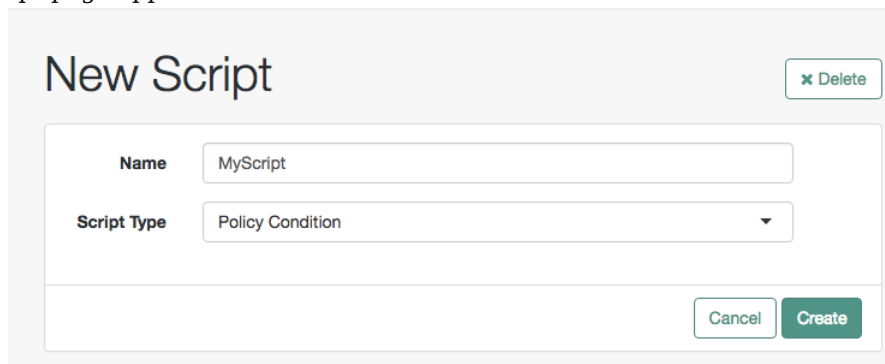
The following procedures describe how to create, modify, and delete scripts using the AM console:

- "To Create Scripts by Using the AM Console"
- "To Modify Scripts by Using the AM Console"
- "To Delete Scripts by Using the AM Console"

To Create Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator. For example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Click New Script.


The New Script page appears:



The screenshot shows a web form titled "New Script". In the top right corner, there is a button labeled "Delete" with a small 'x' icon. The form has two main input fields: "Name" with the text "MyScript" and "Script Type" with a dropdown menu showing "Policy Condition". At the bottom right of the form, there are two buttons: "Cancel" and "Create".

4. Specify a name for the script.
5. Select the type of script from the Script Type drop-down list.
6. Click Create.

The *Script Name* page appears:

 SCRIPT

MyScript

[Delete](#)

Name

Description

Script Type [Change](#)

Language JavaScript Groovy

Script

```
1  /**
2  * This is a Policy Condition example script. It demon
3  * use that information in external HTTP calls and mak
4  */
5
6  var userAddress, userIP, resourceHost;
7
8  if (validateAndInitializeParameters()) {
9
10     var countryFromUserAddress = getCountryFromUserAdd
11     logger.message("Country retrieved from user's addr
12     var countryFromUserIP = getCountryFromUserIP();
13     logger.message("Country retrieved from user's IP:
14     var countryFromResourceURI = getCountryFromResourc
15     logger.message("Country retrieved from resource UR
16
17     if (countryFromUserAddress === countryFromUserIP &
18         logger.message("Authorization Succeeded");
19         responseAttributes.put("countryOfOrigin", {cou
20         authorized = true;
21     } else {
```

[Upload](#) [Validate](#) [Edit Fullscreen](#)

[Save Changes](#)

7. Enter values on the *Script Name* page as follows:

- Enter a description of the script.
- Choose the script language, either JavaScript or Groovy. Note that not every script type supports both languages.
- Enter the source code in the Script field.

On supported browsers, you can click Upload, navigate to the script file, and then click Open to upload the contents to the Script field.

- Click Validate to check for compilation errors in the script.

Correct any compilation errors, and revalidate the script until all errors have been fixed.

- e. Save your changes.

To Modify Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator. For example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Select the script you want to modify from the list of scripts.

The *Script Name* page appears.

4. Modify values on the *Script Name* page as needed. Note that if you change the Script Type, existing code in the script is replaced.
5. If you modified the code in the script, click Validate to check for compilation errors.

Correct any compilation errors, and revalidate the script until all errors have been fixed.

6. Save your changes.

To Delete Scripts by Using the AM Console

1. Log in to the AM console as an AM administrator. For example, `amAdmin`.
2. Navigate to Realms > *Realm Name* > Scripts.
3. Choose one or more scripts to delete by activating the checkboxes in the relevant rows. Note that you can only delete user-created scripts—you cannot delete the global sample scripts provided with AM.
4. Click Delete.

Chapter 4

Managing Scripts (REST)

AM provides the `scripts` endpoint to manage scripts using REST calls.

The following actions are available:

- "Querying Scripts"
- "Reading a Script"
- "Validating a Script"
- "Creating a Script"
- "Updating a Script"
- "Deleting a Script"

User-created scripts are realm-specific, hence the URI for the scripts' API can contain a realm component, such as `/json{/realm}/scripts`. If the realm is not specified in the URI, the top level realm is used.

Tip

AM includes some global example scripts that can be used in any realm.

Scripts are represented in JSON and take the following form. Scripts are built from standard JSON objects and values (strings, numbers, objects, sets, arrays, `true`, `false`, and `null`). Each script has a system-generated *universally unique identifier* (UUID), which must be used when modifying existing scripts. Renaming a script will not affect the UUID:

```
{
  "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
  "name": "Scripted Module - Server Side",
  "description": "Default global script for server side Scripted Authentication Module",
  "script": "dmFyIFNUNUQVJUX1RJ...",
  "language": "JAVASCRIPT",
  "context": "AUTHENTICATION_SERVER_SIDE",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

The values for the fields shown in the example are explained below:

_id

The UUID that AM generates for the script.

name

The name provided for the script.

description

An optional text string to help identify the script.

script

The source code of the script. The source code is in UTF-8 format and encoded into Base64.

For example, a script such as the following:

```
var a = 123;
var b = 456;
```

When encoded into Base64 becomes:

```
dmFyIGEGPSAxMjM7IA0KdmFyIGIyPSA0NTY7
```

language

The language the script is written in - **JAVASCRIPT** or **GROOVY**.

Language Support per Context

Script Context	Supported Languages
POLICY_CONDITION	JAVASCRIPT, GROOVY
AUTHENTICATION_SERVER_SIDE	JAVASCRIPT, GROOVY
AUTHENTICATION_CLIENT_SIDE	JAVASCRIPT
OIDC_CLAIMS	JAVASCRIPT, GROOVY
AUTHENTICATION_TREE_DECISION_NODE	JAVASCRIPT, GROOVY

context

The context type of the script.

Supported values are:

POLICY_CONDITION

Policy Condition

AUTHENTICATION_SERVER_SIDE

Server-side Authentication

AUTHENTICATION_CLIENT_SIDE

Client-side Authentication

Note

Client-side scripts must be written in JavaScript.

OIDC_CLAIMS

OIDC Claims

AUTHENTICATION_TREE_DECISION_NODE

Authentication scripts used by Scripted Tree Decision authentication nodes.

createdBy

A string containing the universal identifier DN of the subject that created the script.

creationDate

An integer containing the creation date and time, in ISO 8601 format.

lastModifiedBy

A string containing the universal identifier DN of the subject that most recently updated the resource type.

If the script has not been modified since it was created, this property will have the same value as **createdBy**.

lastModifiedDate

A string containing the last modified date and time, in ISO 8601 format.

If the script has not been modified since it was created, this property will have the same value as **creationDate**.

Querying Scripts

To list all the scripts in a realm, as well as any global scripts, perform an HTTP GET to the `/json{/realm}/scripts` endpoint with a `_queryFilter` parameter set to `true`.

Note

If the realm is not specified in the URL, AM returns scripts in the top level realm, as well as any global scripts.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts?_queryFilter=true
{
  "result": [
    {
      "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
      "name": "Scripted Policy Condition",
      "description": "Default global script for Scripted Policy Conditions",
      "script": "LyqCiAqIFRoaxMg...",
      "language": "JAVASCRIPT",
      "context": "POLICY_CONDITION",
      "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1433147666269,
      "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1433147666269
    },
    {
      "_id": "7e3d7067-d50f-4674-8c76-a3e13a810c33",
      "name": "Scripted Module - Server Side",
      "description": "Default global script for server side Scripted Authentication Module",
      "script": "dmFyIFNUQVJUX1RJ...",
      "language": "JAVASCRIPT",
      "context": "AUTHENTICATION_SERVER_SIDE",
      "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "creationDate": 1433147666269,
      "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
      "lastModifiedDate": 1433147666269
    }
  ],
  "resultCount": 2,
  "pagedResultsCookie": null,
  "remainingPagedResults": -1
}
```

Supported `_queryFilter` Fields and Operators

Field	Supported Operators
<code>_id</code>	Equals (eq), Contains (co), Starts with (sw)
<code>name</code>	Equals (eq), Contains (co), Starts with (sw)
<code>description</code>	Equals (eq), Contains (co), Starts with (sw)
<code>script</code>	Equals (eq), Contains (co), Starts with (sw)
<code>language</code>	Equals (eq), Contains (co), Starts with (sw)

Field	Supported Operators
context	Equals (eq), Contains (co), Starts with (sw)

Reading a Script

To read an individual script in a realm, perform an HTTP GET using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

Tip

To read a script in the Top Level Realm, or to read a built-in global script, do not specify a realm in the URL.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realm/root/realm/myrealm/scripts/9de3eb62-f131-4fac-a294-7bd170fd4acb
{
  "_id": "9de3eb62-f131-4fac-a294-7bd170fd4acb",
  "name": "Scripted Policy Condition",
  "description": "Default global script for Scripted Policy Conditions",
  "script": "Ly0qCiAqIFRoaxMg...",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1433147666269,
  "lastModifiedBy": "id=dsameuser,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1433147666269
}
```

Validating a Script

To validate a script, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `validate`. Include a JSON representation of the script and the script language, `JAVASCRIPT` or `GR00VY`, in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
  "script": "dmFyIGEGPSAxMjM7dmFyIGIyPSA0NTY7Cg==",
  "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=validate
{
  "success": true
}
```

If the script is valid the JSON response contains a `success` key with a value of `true`.

If the script is invalid the JSON response contains a `success` key with a value of `false`, and an indication of the problem and where it occurs, as shown below:

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
  "script": "dmFyIGEGPSAxMjM7dmFyIGIyPSA0NTY7ID1WQUxJREFUSU90IFNIT1VMRRCBGQULMPQo=",
  "language": "JAVASCRIPT"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=validate
{
  "success": false,
  "errors": [
    {
      "line": 1,
      "column": 27,
      "message": "syntax error"
    }
  ]
}
```

Creating a Script

To create a script in a realm, perform an HTTP POST using the `/json{/realm}/scripts` endpoint, with an `_action` parameter set to `create`. Include a JSON representation of the script in the POST data.

The value for `script` must be in UTF-8 format and then encoded into Base64.

Note

If the realm is not specified in the URL, AM creates the script in the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
--data '{
  "name": "MyJavaScript",
  "script": "dmFyIGEGPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "description": "An example script"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/?_action=create
{
  "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
  "name": "MyJavaScript",
  "description": "An example script",
  "script": "dmFyIGEGPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1436807766258,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1436807766258
}
```

Updating a Script

To update an individual script in a realm, perform an HTTP PUT using the `/json{/realm}/scripts` endpoint, specifying the UUID in both the URL and the PUT body. Include a JSON representation of the updated script in the PUT data, alongside the UUID.

Note

If the realm is not specified in the URL, AM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.1" \
--request PUT \
--data '{
  "name": "MyUpdatedJavaScript",
  "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "description": "An updated example script configuration"
}' \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/0168d494-015a-420f-ae5a-6a2a5c1126af
{
  "_id": "0168d494-015a-420f-ae5a-6a2a5c1126af",
  "name": "MyUpdatedJavaScript",
  "description": "An updated example script configuration",
  "script": "dmFyIGEgPSAxMjM7CnZhciBiID0gNDU2Ow==",
  "language": "JAVASCRIPT",
  "context": "POLICY_CONDITION",
  "createdBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "creationDate": 1436807766258,
  "lastModifiedBy": "id=amadmin,ou=user,dc=openam,dc=forgerock,dc=org",
  "lastModifiedDate": 1436808364681
}
```

Deleting a Script

To delete an individual script in a realm, perform an HTTP DELETE using the `/json{/realm}/scripts` endpoint, specifying the UUID in the URL.

Note

If the realm is not specified in the URL, AM uses the top level realm.

The `iPlanetDirectoryPro` header is required and should contain the SSO token of an administrative user, such as `amAdmin`, who has access to perform the operation.

```
$ curl \
--request DELETE \
--header "iPlanetDirectoryPro: AQIC5..." \
--header "Accept-API-Version: resource=1.1" \
https://openam.example.com:8443/openam/json/realms/root/realms/myrealm/scripts/0168d494-015a-420f-ae5a-6a2a5c1126af
{}
```

Chapter 5

Managing Scripts (ssoadm)

Use the **ssoadm** command's **create-sub-cfg**, **get-sub-cfg**, and **delete-sub-cfg** subcommands to manage AM scripts.

Create an AM script as follows:

1. Create a script configuration file, for example, `/path/to/myScriptConfigurationFile.txt`, containing the following:

```
script-file=/path/to/myScriptFile.js
language=JAVASCRIPT ❶
name=My New Script
context=AUTHENTICATION_SERVER_SIDE ❷
```

- ❶ Possible values for the `language` property are:

- JAVASCRIPT
- GROOVY

- ❷ Possible values for the `context` property are:

- POLICY_CONDITION
- AUTHENTICATION_SERVER_SIDE
- AUTHENTICATION_CLIENT_SIDE
- OIDC_CLAIMS
- AUTHENTICATION_TREE_DECISION_NODE

2. Run the **ssoadm create-sub-cfg** command. The `--datafile` argument references the script configuration file you created in the previous step:

```
$ ssoadm \
create-sub-cfg \
--realm /myRealm \
--adminid amadmin \
--password-file /tmp/pwd.txt \
--servicename ScriptingService \
--subconfigname scriptConfigurations/scriptConfiguration \
--subconfigid myScriptID \
--datafile /path/to/myScriptConfigurationFile.txt
Sub Configuration scriptConfigurations/scriptConfiguration was added to realm /myRealm
```


To list the properties of a script, run the **ssoadm get-sub-cfg** command:

```
$ ssoadm \  
get-sub-cfg \  
--realm /myRealm \  
--adminid amadmin \  
--password-file /tmp/pwd.txt \  
--servicename ScriptingService \  
--subconfigname scriptConfigurations/myScriptID  
createdBy=  
lastModifiedDate=  
lastModifiedBy=  
name=My New Script  
context=AUTHENTICATION_SERVER_SIDE  
description=  
language=JAVASCRIPT  
creationDate=  
script=...Script output follows...
```

To delete a script, run the **ssoadm delete-sub-cfg** command:

```
$ ssoadm \  
delete-sub-cfg \  
--realm /myRealm \  
--adminid amadmin \  
--password-file /tmp/pwd.txt \  
--servicename ScriptingService \  
--subconfigname scriptConfigurations/myScriptID  
Sub Configuration scriptConfigurations/myScriptID was deleted from realm /myRealm
```

Glossary

Access control	Control to grant or to deny access to a resource.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Actions	Defined as part of policies, these verbs indicate what authorized identities can do to resources.
Advice	In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.
Agent administrator	User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent.
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.
Application	<p>In general terms, a service exposing protected resources.</p> <p>In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.</p>
Application type	<p>Application types act as templates for creating policy applications.</p> <p>Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.</p>

	Application types also define the internal normalization, indexing logic, and comparator logic for applications.
Attribute-based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.
Authentication	The act of confirming the identity of a principal.
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.
Authentication level	Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.
Authentication module	AM authentication unit that handles one way of obtaining and verifying credentials.
Authorization	The act of determining whether to grant or to deny a principal access to a resource.
Authorization Server	In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.
Client-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a <i>reference</i> to token to the client.
Client-based sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent

	<p>request. For browser-based clients, AM sets a cookie in the browser that contains the session information.</p> <p>For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.</p>
Conditions	<p>Defined as part of policies, these determine the circumstances under which which a policy applies.</p> <p>Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.</p> <p>Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.</p>
Configuration datastore	LDAP directory service holding AM configuration data.
Cross-domain single sign-on (CDSSO)	AM capability allowing single sign-on across different DNS domains.
CTS-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a <i>reference</i> to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.
CTS-based sessions	AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Delegation	Granting users administrative privileges with AM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given identity in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.
Extended metadata	Federation configuration information specific to AM.
Extensible Access Control Markup Language (XACML)	Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.
Federation	Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and

	allowing principals to access services across different providers without authenticating repeatedly.
Fedlet	Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where AM runs.
Identity	Set of data that uniquely describes a person or a thing such as a device or an application.
Identity federation	Linking of a principal's identity across multiple providers.
Identity provider (IDP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java agent	Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.
Metadata	Federation configuration information for a provider.
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions.
Policy agent	Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.
Policy Administration Point (PAP)	Entity that manages and stores policy definitions.
Policy Decision Point (PDP)	Entity that evaluates access rights and then issues authorization decisions.
Policy Enforcement Point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.
Policy Information Point (PIP)	Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.
Principal	Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities.

	When a Subject successfully authenticates, AM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.
Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.
Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
Authentication Session	The interval while the user or entity is authenticating to AM.
Session	The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions.

Session high availability	Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.
Session token	Unique identifier issued by AM after successful authentication. For a CTS-based sessions, the session token is used to track a principal's session.
Single log out (SLO)	Capability allowing a principal to end a session once, thereby ending her session across multiple applications.
Single sign-on (SSO)	Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.
Site	<p>Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability.</p> <p>The load balancer can also be used to protect AM services.</p>
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	<p>Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user.</p> <p>All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.</p>
Subject	<p>Entity that requests access to a resource</p> <p>When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.</p>
Identity store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom IdRepo implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.