External Services Guide

/ ForgeRock Identity Management 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

Abstract

Guide to configuring external email, and external REST access.

Table of Contents

Overview

This guide covers outbound email, and external REST.

Quick Start

 Email Configure outbound email from IDM.	 External REST Access External REST Services.

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

Configure Outbound Email

The outbound email service sends email from IDM, using a script or the REST API.

You can edit the email service over REST at the `config/external.email` endpoint, or in the `external.email.json` file in your project's `conf` directory.

To configure the outbound email service in the Admin UI, select Configure > Email Settings.

This sample email configuration *(You can edit the email service over REST at the config/external.email endpoint, or in the external.email.json file in your project's conf directory.)* sets up the outbound email service:

+ *Sample Email Configuration*

```
{
    "host" : "smtp.gmail.com",
    "port" : 587,
    "debug" : false,
    "auth" : {
        "enable" : true,
        "username" : "xxxxxxxx",
        "password" : "xxxxxxxx"
    },
    "timeout" : 300000,
    "writetimeout" : 300000,
    "connectiontimeout" : 300000,
    "starttls" : {
        "enable" : true
    },
    "ssl" : {
        "enable" : false
    },
    "smtpProperties" : [
        "mail.smtp.ssl.protocols=TLSv1.2",
        "mail.smtps.ssl.protocols=TLSv1.2"
    ],
    "threadPoolSize" : 20
}
```

1. Edit the email configuration *(You can edit the email service over REST at the config/external.email endpoint, or in the external.email.json file in your project's conf directory.)* to reflect the mail server details and the account that is used to send messages. The complete list of configuration properties is as follows:

   + *External Email Configuration Properties*

`host`

The host name or IP address of the SMTP server. This can be the `localhost`, if the mail server is on the same system as IDM.

`port`

SMTP server port number, such as 25, 465, or 587.

Note

Many SMTP servers require the use of a secure port such as 465 or 587. Many ISPs flag email from port 25 as spam.

`debug`

When set to `true`, this option outputs diagnostic messages from the JavaMail library. Debug mode can be useful if you are having difficulty configuring the external email endpoint with your mail server.

`auth`

The authentication details for the mail account from which emails will be sent.

- `enable`—indicates whether you need login credentials to connect to the SMTP server.

Note

If `"enable" : false,` you can leave the entries for `"username"` and `"password"` empty:

```
"enable" : false,
"username" : "",
"password" : ""
```

- `username`—the account used to connect to the SMTP server.
- `password`—the password used to connect to the SMTP server.

`starttls`

If `"enable" : true`, enables the use of the STARTTLS command (if supported by the server) to switch the connection to a TLS-protected connection before issuing any login commands. If the server does not support STARTTLS, the connection continues without the use of TLS.

`from`

(Optional) Specifies a default `From:` address, that users see when they receive emails from IDM.

`ssl`

Set `"enable" : true` to use SSL to connect, and to use the SSL port by default.

`smtpProperties`

Specifies the SSL protocols that will be enabled for SSL connections. Protocols are specified as a whitespace-separated list. The default protocol is TLSv1.2.

`threadPoolSize`

(Optional) Emails are sent in separate threads managed by a thread pool. This property sets the number of concurrent emails that can be handled at a specific time. The default thread pool size (if none is specified) is `20`.

`connectiontimeout` **(integer, optional)**

The socket connection timeout, in milliseconds. The default connection timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

`timeout` **(integer, optional)**

The socket read timeout, in milliseconds. The default read timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

`writetimeout` **(integer, optional)**

The socket write timeout, in milliseconds. The default write timeout (if none is specified) is `300000` milliseconds, or 5 minutes. A setting of 0 disables this timeout.

2. Submit the configuration over REST or copy the file to your project's `conf/` directory. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--request PUT \
--data '{
    "host" : "smtp.gmail.com",
    "port" : 587,
    "debug" : false,
    "auth" : {
        "enable" : true,
        "username" : "admin",
```

```
        "password" : "Passw0rd"
    },
    "from" : "admin@example.com",
    "timeout" : 300000,
    "writetimeout" : 300000,
    "connectiontimeout" : 300000,
    "starttls" : {
        "enable" : true
    },
    "ssl" : {
        "enable" : false
    },
    "smtpProperties" : [
        "mail.smtp.ssl.protocols=TLSv1.2",
        "mail.smtps.ssl.protocols=TLSv1.2"
    ],
    "threadPoolSize" : 20
}' \
"http://localhost:8080/openidm/config/external.email"
```

Note

IDM encrypts the password.

Send Mail Over REST

Although you are more likely to send mail from a script in production, you can send email using the REST API by sending an HTTP POST to `/openidm/external/email`, to test that your configuration works. You pass the message parameters as part of the POST payload, URL encoding the content as necessary.

The following example sends a test email using the REST API:

```
curl \
 --header "Content-Type: application/json" \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --request POST \
 --data '{
   "from":"openidm@example.com",
   "to":"your_email@example.com",
   "subject":"Test",
   "body":"Test"}' \
 "http://localhost:8080/openidm/external/email?_action=send"
{
  "status": "OK",
  "message": "Email sent"
}
```

By default, a response is returned only when the SMTP relay has completed. To return a response immediately, without waiting for the SMTP relay to finish, include the parameter

`waitForCompletion=false` in the REST call. Use this option only if you do not need to verify that the email was accepted by the SMTP server. For example:

```
curl \
 --header "Content-Type: application/json" \
 --header "X-OpenIDM-Username: openidm-admin" \
 --header "X-OpenIDM-Password: openidm-admin" \
 --header "Accept-API-Version: resource=1.0" \
 --request POST \
 --data '{
   "from":"openidm@example.com",
   "to":"your_email@example.com",
   "subject":"Test",
   "body":"Test"}' \
 "http://localhost:8080/openidm/external/email?_action=send&waitForCompletion=false"
{
  "status": "OK",
  "message": "Email submitted"
}
```

Send Mail From a Script

You can send email by using the resource API functions, with the `external/email` context. For more information about these functions, see "*Scripting Function Reference*" in the *Scripting Guide*. In the following example, `params` is an object that contains the POST parameters.

```
var params =  new Object();
params.from = "openidm@example.com";
params.to = "your_email@example.com";
params.cc = "bjensen@example.com,scarter@example.com";
params.subject = "OpenIDM recon report";
params.type = "text/html";
params.body = "<html><body><p>Recon report follows...</p></body></html>";

openidm.action("external/email", "send", params);
```

IDM supports the following POST parameters.

`from`

Sender mail address

`to`

Comma-separated list of recipient mail addresses

`cc`

Optional comma-separated list of copy recipient mail addresses

`bcc`

Optional comma-separated list of blind copy recipient mail addresses

`subject`

Email subject

`body`

Email body text

`type`

Optional MIME type. One of `"text/plain"`, `"text/html"`, or `"text/xml"`.

Rate Limiting Emails

No rate limiting is applied to password reset emails, or any emails sent by the IDM server. This means that an attacker can potentially spam a known user account with an infinite number of emails, filling that user's inbox. In the case of password reset, the spam attack can obscure an actual password reset attempt.

In a production environment, you must configure email rate limiting through the network infrastructure in which IDM runs. Configure the network infrastructure to detect and prevent frequent repeated requests to publicly accessible web pages, such as the password reset page. You can also handle rate limiting within your email server.

Chapter 2

Access External REST Services

The external REST service lets you access remote REST services at the `openidm/external/rest` context path, or by specifying the `external/rest` resource in your scripts. Note that this service is not intended as a full connector to synchronize or reconcile identity data, but as a way to make dynamic HTTP calls as part of the IDM logic. For more declarative and encapsulated interaction with remote REST services, and for synchronization or reconciliation operations, use the scripted REST implementation of the Groovy connector in the *Connectors Guide*.

An external REST call via a script might look something like the following:

```
openidm.action("external/rest", "call", params);
```

The `call` parameter specifies the action name to be used for this invocation, and is the standard method signature for the `openidm.action` method.

An external REST call over REST might look something like the following:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "url": "http://urlecho.appspot.com/echo?status=200&Content-Type=application%2Fjson&body=%5B%7B%22key
%22%3A%22value%22%7D%5D",
  "method": "GET"
}' \
"http://localhost:8080/openidm/external/rest?_action=call"
[
  {
    "key": "value"
  }
]
```

Invocation Parameters

The following parameters are passed in the resource API parameters map. These parameters can override the static configuration (if present) on a per-invocation basis.

`url`

The target URL to invoke, in string format.

`method`

The HTTP action to invoke, in string format.

Possible actions include `POST`, `GET`, `PUT`, `DELETE`, and `OPTIONS`.

`headers` **(optional)**

The HTTP headers to set, in a map format from string (*header-name*) to string (*header-value*). For example, `Accept-Language: en-US`.

`contentType` **(optional)**

The media type of the data that is sent, for example `"contentType" : "application/json"`. This parameter is applied only if no `Content-Type` header is included in the request. (If a `Content-Type` header is included, that header takes precedence over this `contentType` parameter.) If no `Content-Type` is provided (in the header or with this parameter), the default content type is `application/json; charset=utf-8`.

`body` **(optional)**

The body or resource representation to send (for PUT and POST operations), in string format.

`base64` **(boolean, optional)**

Indicates that the `body` is base64-encoded, and should be decoded prior to transmission.

`forceWrap` **(boolean, optional)**

Indicates that the response must be wrapped in the headers/body JSON message format, even if the response was JSON, and would otherwise have been passed through unchanged.

If you need to disambiguate between HTTP 20x response codes, you must invoke the external REST service with `forceWrap=true`. For failure cases, the HTTP status code is present within the wrapped response embedded in the exception detail, or through the resource exception itself. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--request POST \
--data '{
  "url": "http://urlecho.appspot.com/echo?status=203&Content-Type=application%2Fjson&body=%5B%7B%22key
%22%3A%22value%22%7D%5D",
  "method": "GET",
  "forceWrap": true}' \
"http://localhost:8080/openidm/external/rest?_action=call"
{
  "headers": {
    "Access-Control-Allow-Origin": [
      "*"
    ],
    "Cache-Control": [
```

```
      "max-age=3600"
    ],
    "Content-Length": [
      "17"
    ],
    "Content-Type": [
      "application/json"
    ],
    "Date": [
      "Fri, 17 Jul 2020 10:55:54 GMT"
    ],
    "Server": [
      "Google Frontend"
    ],
    "X-Cloud-Trace-Context": [
      "11e4441659a85832e47af219d6e657af"
    ]
  },
  "code": 203,
  "body": [
    {
      "key": "value"
    }
  ]
}
```

`authenticate`

The authentication type, and the details with which to authenticate.

IDM supports the following authentication types:

- `basic` authentication with a username and password, for example:

```
"authenticate" : {
    "type": "basic",
    "user" : "john",
    "password" : "Passw0rd"
}
```

- `bearer` authentication, with an OAuth token instead of a username and password, for example:

```
"authenticate" : {
    "type": "bearer",
    "token" : "ya29.iQDWKpn8AHy09p....."
}
```

If no `authenticate` parameter is specified, no authentication is used.

Support for Non-JSON Responses

The external REST service supports any arbitrary payload (currently in stringified format). If the response is anything other than JSON, a JSON message object is returned:

- For text-compatible (non-JSON) content, IDM returns a JSON object similar to the following:

```
{
    "headers": { "Content-Type": ["..."] },
    "body": "..."
}
```

- Content that is not text-compatible (such as JPEGs) is base64-encoded in the response `body` and returned as follows:

```
{
    "headers": { "Content-Type": ["..."] },
    "body": "...",
    "base64": true
}
```

Note

If the response format is JSON, the raw JSON response is returned. If you want to inspect the response headers, set `forceWrap` to `true` in your request. This setting returns a JSON message object with `headers` and `body`, similar to the object returned for text-compatible content.

Configure the External REST Service

You can edit the external REST configuration over REST at the `config/external.rest` endpoint, or in an `external.rest.json` file in your project's `conf` directory.

This sample external REST configuration *(You can edit the external REST configuration over REST at the config/external.rest endpoint, or in a conf/external.rest.json file.)* sets up the external REST service:

+ *Sample External REST Configuration*

```
{
  "socketTimeout" : "10 s",
  "connectionTimeout" : "10 s",
  "reuseConnections" : true,
  "retryRequests" : true,
  "maxConnections" : 64,
  "tlsVersion": "&{openidm.external.rest.tls.version}",
  "hostnameVerifier": "&{openidm.external.rest.hostnameVerifier}",
  "proxy" : {
    "proxyUri" : "",
    "userName" : "",
    "password" : ""
  }
}
```

The complete list of configuration properties is as follows:

+ *External REST Configuration Properties*

`socketTimeout` **(string)**

The TCP socket timeout, in seconds, when waiting for HTTP responses. The default timeout is 10 seconds.

`connectionTimeout` **(string)**

The TCP connection timeout for new HTTP connections, in seconds. The default timeout is 10 seconds.

`reuseConnections` **(boolean, true or false)**

Specifies whether HTTP connections should be kept alive and reused for additional requests. By default, connections will be reused if possible.

`retryRequests` **(boolean, true or false)**

Specifies whether requests should be retried if a failure is detected. By default requests will be retried.

`maxConnections` **(integer)**

The maximum number of connections that should be pooled by the HTTP client. At most `64` connections will be pooled by default.

`tlsVersion` **(string)**

The TLS version that should be used for connections.

By default, TLS connections made via the external REST service use TLS version 1.2. In some cases, you might need to specify a different TLS version, for example, if you are connecting to a legacy system that supports an old version of TLS that is not accommodated by the backward-compatibility mode of your Java client. If you need to specify that the external REST service use a different TLS version, uncomment the `openidm.external.rest.tls.version` property towards the end of the `resolver/boot.properties` file and set its value, for example:

```
openidm.external.rest.tls.version=TLSv1.3
```

Valid versions for this parameter include TLSv1.1, TLSv1.2, and TLSv1.3.

`hostnameVerifier` **(string)**

Specifies whether the external REST service should check that the hostname to which an SSL client has connected is allowed by the certificate that is presented by the server.

The property can take the following values:

- `STRICT` - hostnames are validated

- `ALLOW_ALL` - the external REST service does not attempt to match the URL hostname to the SSL certificate Common Name, as part of its validation process

By default, this property is set in the `resolver/boot.properties` file and the value in `conf/external.rest.json` references that setting. For testing purposes, the default setting in `boot.properties` is:

```
openidm.external.rest.hostnameVerifier=ALLOW_ALL
```

If you do not set this property (by removing it from the `boot.properties` file or the `conf/external.rest.json` file), the behavior is to validate hostnames (the equivalent of setting `"hostnameVerifier": "STRICT"`). In production environments, you *should* set this property to `STRICT`.

`proxy`

Lets you set a proxy server *specific* to the external REST service. If you set a `proxyUri` here, the system-wide proxy settings described in "HTTP Clients" in the *Setup Guide* are ignored. To configure a system-wide proxy, leave these `proxy` settings empty and configure the HTTP Client settings instead.

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.