

Confidential Clients

Overview

Mobile applications can utilize the MobileFirst client SDKs to request access to protected resources. Other entities which are not mobile applications can do so as well. Such entities are considered as **Confidential Clients**.

Confidential clients are clients that are capable of maintaining the confidentiality of their authentication credentials. You can use the MobileFirst authorization server to grant confidential clients access to protected resources, in accordance with the OAuth specification. This feature allows you to grant access to your resources to non-mobile clients, such as performance-testing applications, and any other kind of back-end that might need to request a protected resource, or use one of the MobileFirst Foundation **REST APIs**, such as the REST API for **push notifications**.

You begin by registering a confidential client with MobileFirst Server. As part of the registration, you provide the credentials of the confidential client, which consist of an ID and a secret. In addition, you set the client's allowed scope, which determines the scopes that can be granted to this client. When a registered confidential client requests an access token from the authorization server, the server authenticates the client by using the registered credentials, and verifies that the requested scope matches the client's allowed scope.

Registered confidential clients can obtain a token to be used in all requests to the MobileFirst Server. This flow is based on the client credentials flow (<https://tools.ietf.org/html/rfc6749#section-1.3.4>) of the OAuth specification.

Registering the confidential client

In the navigation sidebar of the MobileFirst Operations Console, click **Runtime Settings** → **Confidential Clients**. Click **New** to add a new entry.

You must provide the following information:

- **Display Name:** an optional display name that is used to refer to the confidential client. The default display name is the value of the ID parameter. For example: **Back-end Node server**.
- **ID:** A unique identifier for the confidential client (can be considered as a "user name").
- **Secret:** A private passphrase to authorize access from the confidential client (can be considered as an API key).
- **Allowed Scope:** A confidential client that uses such ID and Secret combination is automatically granted the scope that is defined here. Learn more about **Scopes** in the Authorization Concepts ([../#scope](#)) tutorial.
 - An element of an allowed scope can also include the special asterisk wildcard character (*), which signifies any sequence of zero or more characters. For example, if the scope element is `send*`, the confidential client can be granted access to scopes that contain any scope element that starts with "send", such as "sendMessage". The asterisk wildcard can be placed at any position within the scope element, and can also appear more than once.
 - An allowed-scope parameter that consists of a single asterisk character (*) indicates that the confidential client can be granted a token for any scope.

Examples of scopes:

- Protecting external resources (`../protecting-external-resources`) uses the scope `authorization.introspect`.
- Sending a Push Notification (`../notifications/sending-push-notifications`) via the REST API uses the space-separated scope elements `messages.write` and `push.application.<applicationId>`.
- Adapters can be protected by a custom scope element, such as `accessRestricted`.
- The scope `*` is a catch-all scope, granting access to any requested scope.

MobileFirst Operations Console

Home > mfp > Runtime Settings

Runtime Settings

The confidential client was saved successfully.

Runtime Properties Keystore Confidential Clients

Confidential Clients

By using IBM MobileFirst™ Platform Foundation, you can let a confidential (or non-mobile) client connect to mobile services in a secure way. For example, you can grant a back-end service access to the push service.

Client ID	Display Name	Client Secret	Allowed Scope	Actions
test	Test Client	*****	*	
admin	admin	*****	push.* mfp.admin.plugins	
Push	Push	*****	push.application.com.sample.PushNotificationsSwift messages.write	

New

Predefined confidential clients

The MobileFirst Platform Server comes with some predefined confidential clients:

test

The `test` client is available only in development mode. It allows you to test your resources easily.

- **ID:** `test`
- **Secret:** `test`
- **Allowed Scope:** `*` (any scope)

admin

The `admin` client is used internally by the MobileFirst Foundation administration service.

push

The `push` client is used internally by the MobileFirst Foundation push service.

Obtaining an access token

A token can be obtained from the MobileFirst Server **token endpoint**.

For **testing purposes**, you can use Postman as described below.

In a real situation, implement Postman in your back-end logic, with the technology of your choice.

1. Make a **POST** request to **http(s)://[ipaddress-or-hostname]:[port]/[runtime]/api/az/v1/token**.
For example: `http://localhost:9080/mfp/api/az/v1/token`
 - In a development environment, the MobileFirst Server uses a pre-existing `mfp` runtime.
 - In a production environment, replace the runtime value with your runtime name.
2. Set the request with a content-type of `application/x-www-form-urlencoded`.
3. Set the following two form parameters:
 - `grant_type`: `client_credentials`
 - `scope`: Use the scope protecting the resource.If you don't use a scope to protect your resource, use an empty string.

The screenshot shows the Postman Builder interface for a POST request to `http://localhost:9080/mfp/api/az/v1/token`. The 'Body' tab is selected, and the content type is set to `x-www-form-urlencoded`. Two form parameters are defined:

key	value
<code>grant_type</code>	<code>client_credentials</code>
<code>scope</code>	<code>messages.write push.application.com.sample.PushNotificationsAndroid</code>

4. To authenticate the request, use Basic Authentication (https://en.wikipedia.org/wiki/Basic_access_authentication#Client_side). Use your confidential client's **ID** and **Secret**.

The screenshot shows the Postman Builder interface with the 'Authorization' tab selected. The 'Type' is set to 'Basic Auth'. The 'Username' field contains 'test' and the 'Password' field contains a masked value. A checkbox for 'Show Password' is checked. A note states: 'The authorization header will be generated and added as a custom header'. There are 'Clear' and 'Update Request' buttons.

Outside of Postman, if you use the **test** confidential client, set the **HTTP header** to `Authorization: Basic dGVzdDp0ZXN0` (`test:test` encoded using **base64**).

The response for this request will contain a `JSON` object, including the **access token** and its expiration time (1 hour).

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsImp...",
  "token_type": "Bearer",
  "expires_in": 3599,
  "scope": "sendMessage accessRestricted"
}
```



Using the access token

From here on, you can make requests to the desired resources by adding the **HTTP header**:

`Authorization: Bearer eyJhbGciOiJSUzI1NiIsImp3ayI6eyJ1Ijo...`, replacing the access token by the one you extracted from the previous JSON object.

Possible responses

In addition to the normal responses that your resource may generate, look out for a few responses that are generated by MobileFirst Server.

Bearer

An HTTP **401** response status with the HTTP header `WWW-Authenticate : Bearer` means that no token was found on the `Authorization` header of the original request.

invalid_token

An HTTP **401** response status with the HTTP header `WWW-Authenticate: Bearer error="invalid_token"` means that the token that was sent is **invalid** or **expired**.

insufficient_scope

An HTTP **403** response status with the HTTP header `WWW-Authenticate : Bearer error="insufficient_scope", scope="scopeA scopeB"` means that the token found in the original request did not match the **scope required by this resource**. The header also includes the scope it expected.

When making a request, if you do not know which scope is required by the resource, `insufficient_scope` is the way to determine the answer.

For example, request a token with an empty string ("") as the scope value and make a request to the resource. Then, you can extract the required scope from the 403 response and request a new token for this scope.