



MyID PIV
Version 12.10

MyID Authentication Guide

Lutterworth Hall, St Mary's Road, Lutterworth, Leicestershire, LE17 4PS, UK
www.intercede.com | info@intercede.com | [@intercedemyid](https://twitter.com/intercedemyid) | +44 (0)1455 558111

Copyright

© 2001-2024 Intercede Limited. All rights reserved.

Information in this document is subject to change without notice. The software described in this document is furnished exclusively under a restricted license or non-disclosure agreement. Copies of software supplied by Intercede Limited may not be used resold or disclosed to third parties or used for any commercial purpose without written authorization from Intercede Limited and will perpetually remain the property of Intercede Limited. They may not be transferred to any computer without both a service contract for the use of the software on that computer being in existence and written authorization from Intercede Limited.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or any means electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Intercede Limited.

Whilst Intercede Limited has made every effort in the preparation of this manual to ensure the accuracy of the information, the information contained in this manual is delivered without warranty, either express or implied. Intercede Limited will not be held liable for any damages caused, or alleged to be caused, either directly or indirectly by this manual.

Licenses and Trademarks

The Intercede® and MyID® word marks and the MyID® logo are registered trademarks of Intercede in the UK, US and other countries.

Microsoft and Windows are registered trademarks of Microsoft Corporation. Other brands and their products are trademarks or registered trademarks of their respective holders and should be noted as such. All other trademarks acknowledged.

Apache log4net

Copyright 2004-2021 The Apache Software Foundation

This product includes software developed at

The Apache Software Foundation (<https://www.apache.org/>).

Conventions used in this document

- Lists:
 - Numbered lists are used to show the steps involved in completing a task when the order is important.
 - Bulleted lists are used when the order is unimportant or to show alternatives.
- **Bold** is used for menu items and for labels.
For example:
 - Record a valid email address in '**From**' email address.
 - Select **Save** from the **File** menu.
- *Italic* is used for emphasis:
For example:
 - Copy the file *before* starting the installation.
 - Do *not* remove the files before you have backed them up.
- ***Bold and italic*** hyperlinks are used to identify the titles of other documents.
For example: "See the ***Release Notes*** for further information."
Unless otherwise explicitly stated, all referenced documentation is available on the product installation media.
- A `fixed width` font is used where the identification of spaces is important, including filenames, example SQL queries and any entries made directly into configuration files or the database.
- **Notes** are used to provide further information, including any prerequisites or configuration additional to the standard specifications.
For example:
Note: This issue only occurs if updating from a previous version.
- **Warnings** are used to indicate where failure to follow a particular instruction may result in either loss of data or the need to manually configure elements of the system.
For example:
Warning: You must take a backup of your database before making any changes to it.

Contents

MyID Authentication Guide	1
Copyright	2
Conventions used in this document	3
Contents	4
1 Introduction	6
2 Setting up the standalone authentication service	7
2.1 Installing the standalone authentication service	8
2.2 Configuring the standalone authentication service	8
2.3 Logging the standalone authentication service	10
3 MyID AD FS Adapter OAuth	11
3.1 Overview	11
3.2 AD FS Adapter OAuth prerequisites	12
3.3 Setting up the ADFS Auth web service	12
3.3.1 Installing the ADFS Auth web service	13
3.3.2 Configuring the ADFS Auth web service	13
3.3.3 Configuring the AD FS server to communicate with the ADFS Auth web service	14
3.3.4 Logging the ADFS Auth web service	14
3.4 Configuring the standalone authentication service for AD FS	14
3.4.1 Generating a shared secret	15
3.4.2 Updating the web.oauth2.ext configuration file	15
3.5 Installing the AD FS Adapter OAuth	18
3.5.1 Uninstalling the AD FS Adapter OAuth	22
3.6 Managing the AD FS Adapter OAuth	23
3.6.1 Configuration file	23
3.6.2 Encrypting the client secret	24
3.6.3 Managing themes	25
3.6.4 Logging for the AD FS Adapter OAuth	25
3.7 Troubleshooting	26
4 Authenticating using OpenID Connect	30
4.1 Configuring the web service for OpenID Connect	32
4.1.1 Creating a shared secret	33
4.1.2 Editing the configuration file	34
4.1.3 Configuring authentication to skip the MyID Authentication screen	41
4.2 Obtaining an identity token	42
4.2.1 Generating a PKCE code verifier and code challenge	42
4.2.2 Requesting an authorization code	43
4.2.3 Requesting an identity token	49
4.3 Troubleshooting	50
5 Authenticating for embedded Operator Client screens	53
5.1 Configuring web.oauth2 for user-based authentication	53
5.2 Obtaining an access token	54
5.2.1 Example requests	55
5.3 Posting the access token	57

6 Setting up an external identity provider	59
6.1 Configuring Microsoft Entra	61
6.1.1 Configuring Microsoft Entra as an external identity provider	61
6.1.2 Encrypting the client secret	62
6.1.3 Configuring the web.oauth2 server for Microsoft Entra	63
6.1.4 Configuring MyID to use Microsoft Entra	65
6.1.5 Next steps	65
6.2 Configuring OpenID Connect	66
6.2.1 Configuring OpenID Connect as an external identity provider	66
6.2.2 Encrypting the client secret	67
6.2.3 Configuring the web.oauth2 server for OpenID Connect	68
6.2.4 Configuring MyID to use OpenID Connect	71
6.2.5 Next steps	71
6.3 Configuring other types of identity provider	72
6.4 Mapping attributes	73
6.4.1 Finding a list of available claims	73
6.4.2 Matching attributes	74
6.4.3 Mapping attributes	75
6.5 Example Microsoft Entra settings	79
6.6 Example OpenID Connect settings	81
6.6.1 OpenID Connect settings for Microsoft Entra	81
6.6.2 OpenID Connect settings for Okta	83
7 Reporting on the authentication database	85

1 Introduction

The MyID[®] authentication service (web.oidc) provides authentication services for the MyID Operator Client and the MyID Core API.

You can also use the authentication service in the following ways:

- Using a standalone version of the authentication service (web.oidc.ext) for high availability operations.

See section 2, *Setting up the standalone authentication service*.

- With an ADFS adapter for FIDO authenticators using OAuth.

See section 3, *MyID ADFS Adapter OAuth*.

- Using OpenID to provide authentication for an external system.

See section 4, *Authenticating using OpenID Connect*.

To assist you in using these features, the MyID authentication service provides the following:

- Reporting on audited actions stored in the authentication database.

See section 7, *Reporting on the authentication database*.

2 Setting up the standalone authentication service

The standard web.oidc MyID authentication service (web.oidc) runs on the MyID web server with the other MyID web services, and communicates with the MyID database through the MyID application server. This places a reliance on the entire MyID system; for a mission-critical authentication service, providing access to crucial systems, you may need to use a standalone version of this service – web.oidc.ext – that does not rely on the MyID infrastructure.

For example, the MyID AD FS Adapter OAuth relies on the standalone version of the web service instead of the standard MyID authentication service; see section 3, [MyID AD FS Adapter OAuth](#) for details. You can choose to use either the standard or standalone versions of the web service when obtaining an identity token using OpenID; see section 4, [Authenticating using OpenID Connect](#).

The standalone authentication service communicates directly with the MyID main and authentication databases; as the service is used for authentication and not registration, it needs read-only access to the main MyID database, and needs read-write access only for the authentication database.

Note: Currently, the standalone authentication service supports only the FIDO method authentication; you cannot use the standalone authentication service to provide authentication using MyID security phrases, smart cards, or authentication codes, for example.

This section contains information on:

- Installing the web service.
See section 2.1, [Installing the standalone authentication service](#).
- Configuring the web service.
See section 2.2, [Configuring the standalone authentication service](#).
- Logging the web service.
See section 2.3, [Logging the standalone authentication service](#).

2.1 Installing the standalone authentication service

You can install the standalone authentication web service on the MyID web server, or on its own dedicated web server. The requirements for a dedicated web server are the same as for the main MyID web server; in addition, the dedicated web server must be able to communicate with the MyID main and authentication databases; while the `web.oauth2` web service communicates with the database through the MyID application server, the `web.oauth2.ext` web service removes this reliance on the application server and communicates directly with the databases.

You are recommended to install your MyID system and configure it before you install the standalone web service.

Because the standalone authentication web service communicates directly with the database rather than going through the application server, you do not need to install COM+ proxies, but you do need to install the following database connectivity software:

- Microsoft OLE DB Driver 19 for SQL Server (MSOLEDBSQL).

This driver is available from Microsoft.

For more information about the Microsoft OLE DB Driver, see the *Installing the database software* section in the [Installation and Configuration Guide](#).

- SQL Server Native Client 11

This is available in the SQL Server Feature Pack.

To install the standalone authentication service, run the MyID Installation Assistant, and on the Server Roles and Features screen, select only the **External Authentication Server > web.oauth2.ext** option.

For more information, see the *Selecting the server roles and features* section in the [Installation and Configuration Guide](#).

2.2 Configuring the standalone authentication service

When you install the standalone authentication service, the installation program configures the service with the database connection details for your main MyID database and the MyID authentication database. These settings are initially stored in the `appsettings.json` file. If you need to update the database connection details, for example if you are using SQL Authentication and the password has changed, you can edit the `appsettings.Production.json` file to override the settings in the `appsettings.json` file.

Note: If you subsequently install or upgrade MyID again and provide different database connection information in the MyID installation program, and you have set the `ConnectionStringCore` or `ConnectionStringAuth` options in the `appsettings.Production.json` file, the values you enter in the installation program are ignored; the `appsettings.Production.json` file is never updated by the installation program, and always takes precedence over the `appsettings.json` file.

To edit the database connection strings:

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program  
Files\Intercede\MyID\web.oauth2.ext\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service.

2. Set the following options in the `MyID:Database` section:

- `ConnectionStringCore` – contains the connection string for the main MyID database.
- `ConnectionStringAuth` – contains the connection string for the MyID authentication database.

You can copy the original details from the `appsettings.json` file if necessary.

Your `appsettings.Production.json` file may already contain commented-out entries for these values; remove the double-slash `//` to uncomment the entries.

3. If you need to update the password in the `appsettings.json` file, you can use the Password Change Tool; see the *Working with SQL accounts* section in the [Password Change Tool](#) guide.

Alternatively, if you want to edit the settings in the `override appsettings.Production.json` file, you must update the password manually:

- a. Log on to the server as the MyID Authentication user.

This is the user under which the standalone authentication service runs – you can check the identity used for the **myid.web.oauth2.ext.pool** application pool to confirm.

- b. Open a Windows PowerShell command prompt, and navigate to the `web.oauth2.ext` web service folder.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2.ext\
```

- c. Run the following PowerShell script:

```
.\DPAPIEncrypt.ps1 <password>
```

For example:

```
.\DPAPIEncrypt.ps1 mypassword1234
```

The script outputs an encrypted copy of your new password; for example:

```
PS C:\Program Files\Intercede\MyID\web.oauth2.ext>
.\DPAPIEncrypt.ps1 mypassword1234
AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAA7X [...]
cJ0kGfzCRQAAApCVkhSoyCs4xotykdKZ3w9gitg==
```

(Encrypted output string truncated for documentation purposes.)

- d. Copy the encrypted password, then add it to the `PasswordDPAPI` field in the connection string.

For example:

```
"ConnectionStringCore": "Database=MyID; Server=myserver.example.com;
User Id=sa; PasswordDPAPI=AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAA7X [...]
cJ0kGfzCRQAAApCVkhSoyCs4xotykdKZ3w9gitg==";
```

4. Save the `appsettings.Production.json` file.
5. Recycle the web service app pool:
 - a. On the MyID web server, in Internet Information Services (IIS) Manager, select **Application Pools**.
 - b. Right-click the **myid.web.oauth2.ext.pool** application pool, then from the pop-up menu click **Recycle**.

This ensures that the web service has picked up the changes to the configuration file.

2.3 Logging the standalone authentication service

You can configure logging for the web service; see the *MyID REST and authentication web services* section in the [Configuring Logging](#) guide for details.

3 MyID AD FS Adapter OAuth

The AD FS Adapter OAuth provides Active Directory Federation Services (AD FS) with FIDO2 authentication mechanism, using a FIDO2 security key to allow identity access to a Relying Party Trust. You can configure the AD FS Adapter OAuth as a primary or additional authentication step using the AD FS Manager Tool.

You must deploy the AD FS Adapter OAuth to your AD FS servers. You can install the AD FS Adapter OAuth on a system running Windows Server 2016 or 2019; however, only Windows Server 2019 supports the use of additional authentication methods as primary authenticators.

This section contains the following information:

- An overview of the AD FS authentication process.
See section [3.1, Overview](#).
- AD FS prerequisites.
See section [3.2, AD FS Adapter OAuth prerequisites](#).
- Information on installing and configuring the ADFS Auth web service.
See section [3.3, Setting up the ADFS Auth web service](#).
- Details on using the standalone authentication service for AD FS authentication.
See section [3.4, Configuring the standalone authentication service for AD FS](#).
- Instructions for installing the AD FS Adapter OAuth plug-in on the AD FS server.
See section [3.5, Installing the AD FS Adapter OAuth](#).
- Information on configuration options and themes for the AD FS Adapter OAuth.
See section [3.6, Managing the AD FS Adapter OAuth](#).
- Troubleshooting issues with the AD FS Adapter OAuth.
See section [3.7, Troubleshooting](#).

3.1 Overview

The AD FS authentication process has the following components:

- AD FS Adapter OAuth – a plug-in for the AD FS server that provides access to the MyID authentication system.
- ADFS Auth web service – an intermediary web service used to store information for the AD FS Adapter OAuth.
- Standalone authentication web service (web.oauth2.ext) – a standalone version of the MyID authentication web service (web.oauth2) which is used to take the user through the authentication process using their FIDO device.

With the AD FS Adapter OAuth installed and configured on AD FS, providing either primary or additional authentication for a Relying Party Trust, a user starts the authentication process by trying to access the Relying Party Trust service when they enter their email address at the login screen.

AD FS asks the AD FS Adapter OAuth if the email address provided is one it recognizes. The AD FS Adapter OAuth currently assumes all email addresses passed to it are acceptable. AD FS starts the authentication process, calling into the AD FS Adapter OAuth and passing in the claim containing the user's email address.

The AD FS Adapter OAuth then starts the authentication process by displaying a web page, which posts the information to the ADFS Auth web service that will be needed to complete the authentication process with AD FS. The ADFS Auth web service stores this information, whereupon the AD FS Adapter OAuth requests an authorization from the web.oauth2.ext standalone authentication web service.

The standalone authentication web service then takes the user through the authentication process; the user confirms their identity using their registered FIDO authenticator. The authentication web service then redirects to the ADFS Auth web service, passing it the result of the authentication process. The ADFS Auth web service retrieves the information it stored at the start of the process, then passes it back to the AD FS Adapter OAuth along with the result of the authentication process. If the authentication process successfully provided an authorization code, the AD FS Adapter OAuth uses this to obtain an identity token from the standalone authentication web service.

The token is then validated. If validation succeeds, the claims required for a successful AD FS authentication are returned by the AD FS Adapter OAuth to AD FS, and AD FS allows access to the Relying Party Trust.

3.2 AD FS Adapter OAuth prerequisites

The following prerequisites must be in place before you install the AD FS Adapter OAuth:

- Relying Party Trust

A Relying Party Trust must exist under **AD FS Management > AD FS > Relying Party Trusts** to add the AD FS Adapter as a primary or additional authentication method.

- Access Control Policy

A suitable access control policy for controlling access to the Relying Party Trust under **AD FS Management > AD FS > Access Control Policies**; for example, "Permit everyone and require MFA" if the AD FS Adapter is used as an additional authentication method, or "Permit everyone" if the AD FS Adapter is used as a primary authentication method.

- AD FS Service Account

The AD FS service account must be a member of the "domain users". The AD FS service account needs "log on as a service" permission. To set this option, from **AD FS Server Manager > Tools > Local Security Policy > Security Settings > Local Policies > User Rights Assignment > Log on as a service > Local Security Setting tab > Add User or Group**, add the AD FS service account user.

3.3 Setting up the ADFS Auth web service

The ADFS Auth web service is an intermediary web service used to store information for the AD FS Adapter OAuth. You can install it on the same server as the standalone authentication service (web.oauth2.ext).

For information on how the ADFS Auth web service fits into the AD FS authentication architecture, see section [3.1, Overview](#).

3.3.1 Installing the ADFS Auth web service

To install the standalone authentication service, run the MyID Installation Assistant, and on the Server Roles and Features screen, select **External Authentication Server > AD FS Auth Web Service** option.

You can install this service on the same server as the standalone authentication service (web.oauth2.ext).

For more information about running the MyID Installation Assistant, see the *Selecting the server roles and features* section in the [Installation and Configuration Guide](#).

3.3.2 Configuring the ADFS Auth web service

Once you have installed the ADFS Auth web service, you must configure the service with the location of your AD FS. This allows the ADFS Auth web service to accept connections from the AD FS Adapter OAuth installed on the AD FS server.

To configure the ADFS Auth web service:

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program Files\Intercede\MyID\AdfsAuth\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service. If this file does not already exist, you must create it in the same folder as the `appsettings.json` file.

2. Set the following:

```
"AllowedOrigins": [ "https://<ADFS domain>" ]
```

where `<ADFS domain>` is the domain of your AD FS server; for example:

```
"AllowedOrigins": [ "https://adfs.example.com" ]
```

3. Save the `appsettings.Production.json` file.
4. Recycle the web service app pool:
 - a. On the MyID web server, in Internet Information Services (IIS) Manager, select **Application Pools**.
 - b. Right-click the **AdfsAuthPool** application pool, then from the pop-up menu click **Recycle**.

This ensures that the web service has picked up the changes to the configuration file.

3.3.3 Configuring the AD FS server to communicate with the ADFS Auth web service

On the AD FS server, you must configure AD FS to set the `Content-Security-Policy` to allow it to http POST to the domain on which the ADFS Auth web service runs.

For example, run the following PowerShell commands:

```
Set-AdfsResponseHeaders -SetHeaderName "Content-Security-Policy" -SetHeaderValue "default-src 'self' <domainOfAdfsAuthWS> 'unsafe-inline' 'unsafe-eval'; img-src 'self' data;";  
net stop adfssrv  
net start adfssrv
```

where `<domainOfAdfsAuthWS>` is the web domain on which the ADFS Auth web service runs.

3.3.4 Logging the ADFS Auth web service

You can configure logging for the web service; see the *MyID REST and authentication web services* section in the [Configuring Logging](#) guide for details.

3.4 Configuring the standalone authentication service for AD FS

The standalone authentication service (`web.oauth2.ext`) provides a connection to the MyID database and allows the AD FS Adapter to authenticate to AD FS using credentials issued using MyID. You must configure the `web.oauth2.ext` service with a shared secret to secure the connection between the adapter and the service. You must also add the URL of the ADFS Auth web service to the list of allowed redirect URLs.

You must make sure that the standalone authentication service is installed. For more information about the `web.oauth2.ext` standalone authentication service, see section 2, [Setting up the standalone authentication service](#).

If `web.oauth2.ext` (which is authenticating the user) is on a different server from `web.oauth2` (which registered the FIDO credential), ensure there is a load balancer or reverse proxy in front of both of these servers, so that the client computer sees the same domain in the URL for both of these machines.

3.4.1 Generating a shared secret

MyID uses a shared secret to secure the connection between the AD FS Adapter and the standalone authentication service.

Important: You must keep the shared secret safe, as it can provide access to the features of the MyID authentication service, so must be an unguessable value; for this reason, you are recommended to generate a GUID for the shared secret. Do not use the same shared secret for other purposes such as end-user authentication or server-to-server API authentication.

You can use the provided `GenClientSecret.ps1` PowerShell script to generate a GUID and create a SHA-256 hash converted to Base64:

1. On the web server, open a Windows PowerShell command prompt.
2. Navigate to the `web.oauth2.ext` folder.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2.ext
```

3. Run the script:

```
.\GenClientSecret.ps1
```

The script generates a GUID to use as the shared secret, creates a SHA-256 hash from the GUID, then converts the hash to Base64; for example:

```
PS C:\Program Files\Intercede\MyID\web.oauth2> .\GenClientSecret.ps1
client secret: 82564d6e-c4a6-4f64-a6d4-cac43781c67c
SHA256+base64: kv31VP5z/oKS0QMMaIfz2UrhmqOdgAPpXV/vaF1cymk=
```

Important: Do not use this example secret in your own system.

4. Take a note of the following values:
 - `client secret` – you need this value for the AD FS Client Secret screen in the AD FS Adapter installation program.
See section [3.5, Installing the AD FS Adapter OAuth](#).
 - `SHA256+base64` – you need this value for the `ClientSecrets` parameter in the `web.oauth2.ext` web service configuration file. The server does not store the secret, only the Base64-encoded hash.

See section [3.4.2, Updating the web.oauth2.ext configuration file](#).

Note: If you are going to add the secret to the AD FS Adapter configuration file manually rather than using the installation program (for example, if you have an installed system, and you want to update the client secret periodically) you are strongly recommended to encrypt the secret using the provided `DPAPIEncrypt.ps1` PowerShell script before adding it to the configuration file; see section [3.6.2, Encrypting the client secret](#) for details. If you use the installation program, the secret is encrypted automatically.

3.4.2 Updating the web.oauth2.ext configuration file

You must update the `web.oauth2.ext` configuration file to include the secret you have generated to secure the connection between the AD FS Adapter and the standalone authentication service, and add the URL of the ADFS Auth web service to the list of allowable redirect URIs.

To configure the `web.oauth2.ext` service:

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program
Files\Intercede\MyID\web.oauth2.ext\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service.

You must edit the `myid.adfs` client section in this configuration file, and provide the required configuration settings.

2. Edit the following section in the `appsettings.Production.json` file:

```
"Clients":[
  {},
  {},
  {
    "ClientId":"myid.adfs",
    "ClientName":"MyID ADFS",
    "ClientSecrets":[
      {
        "Value":"<secret>"
      }
    ],
    "RedirectUri":[
      "https://<auth service domain>/AdfsAuth/home/AdfsAuth"
    ],
    "Properties":
    {
      "Skin": "popover",
      "EnableFido2LoginBasicAssurance": false,
      "EnableFido2LoginHighAssurance": true
    }
  }
]
```

Important: When you have clients in the `appsettings.json` file *and* the `appsettings.Production.json` file, you must make sure the Production file does not overwrite the entries in the base file. In these settings files, entries in arrays are determined by their index; therefore if you have two client entries in the `appsettings.json` file before the `myid.adfs` entry, you must include two blank array entries `{}`, in the `appsettings.Production.json` file before you include the details for the `myid.adfs` client.

3. Set the following values:

- `<secret>` – set this to the Base64-encoded SHA-256 hash of the secret you created; for example:

```
"Value":"kv31VP5z/oKS0QMMaIfZ2Urhmq0dgAPpXV/vaF1cymk="
```

See section [3.4.1, Generating a shared secret](#).

- `<auth service domain>` – set this to the domain of the MyID ADFS Auth server; for example:

```
"https://myserver.example.com/AdfsAuth/home/AdfsAuth"
```


You can set the following Properties:

- `Skin` – currently only `popover` is available as a skin option for the authentication GUI. The authentication web page shows a background image with a popover representing the authentication UI; this is suitable for large windows. If you are displaying the authentication page in a small popup window, you can omit this option.
- `EnableFido2LoginBasicAssurance` – set this to `true` to allow authentication using FIDO basic assurance authenticators, which provide only single-factor authentication.
- `EnableFido2LoginHighAssurance` – set this to `true` to allow authentication using FIDO high assurance authenticators, which provide multi-factor authentication.

When using FIDO with `web.oidc`, authentication using high and basic assurance is controlled using the `logon mechanisms` feature; as the `web.oidc.ext` standalone authentication service does not use the MyID application server, this feature is not available, and you must specify the `logon mechanisms` using the above options.

For more information about basic and high assurance FIDO authenticators, see the [Supported authenticators](#) section in the [FIDO Authenticator Integration Guide](#).

By default, `EnableFido2LoginBasicAssurance` is set to `false`, to prevent authentication with FIDO basic assurance authenticators, and

`EnableFido2LoginHighAssurance` is set to `true`, to allow authentication with FIDO high assurance authenticators.

For example:

```
"Clients":[
  {},
  {},
  {
    "ClientId":"myid.adfs",
    "ClientName":"MyID ADFS",
    "ClientSecrets":[
      {
        "Value":"kv31VP5z/oKS0QMMaIfZ2Urhmq0dgAPpXV/vaF1cymk="
      }
    ],
    "RedirectUris":[
      "https://myserver.example.com/AdfsAuth/home/AdfsAuth"
    ],
    "Properties":
    {
      "Skin": "popover",
      "EnableFido2LoginBasicAssurance": false,
      "EnableFido2LoginHighAssurance": true
    }
  }
]
```

4. Save the `appsettings.Production.json` file.

5. Recycle the web service app pool:
 - a. On the MyID web server, in Internet Information Services (IIS) Manager, select **Application Pools**.
 - b. Right-click the **myid.web.oauth2.ext.pool** application pool, then from the pop-up menu click **Recycle**.

This ensures that the web service has picked up the changes to the configuration file.

3.5 Installing the AD FS Adapter OAuth

Before you install the AD FS Adapter OAuth, make sure you have the following web services installed and configured:

- ADFS Auth web service – see section 3.3, *Setting up the ADFS Auth web service*.
- Standalone authentication service (web.oauth2.ext) – see section 3.4, *Configuring the standalone authentication service for AD FS*.

You must install the AD FS Adapter OAuth on the AD FS server.

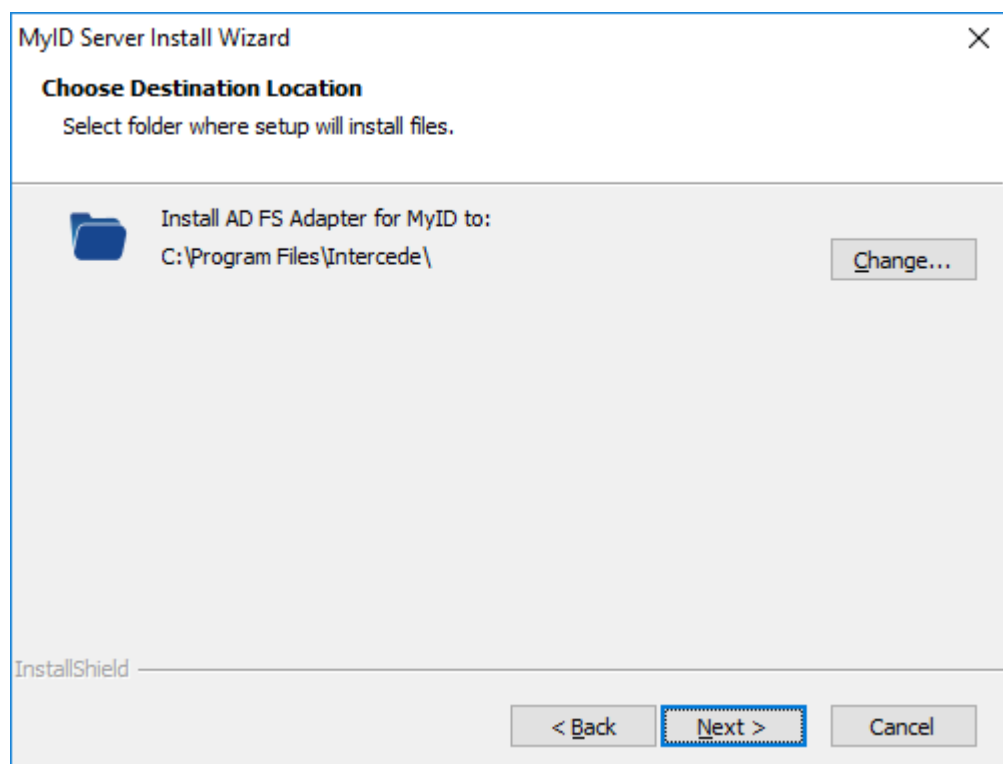
To install the adapter:

1. Copy the installation program onto the AD FS server.

The AD FS Adapter installation program is provided with the MyID installation media in the following folder:

```
\Authentication\AD FS Adapter for MyID\
```

2. Run the .msi installation program.
3. Click **Next** to begin.



4. Select the location for the AD FS Adapter.

By default, the AD FS Adapter is installed to the following location:

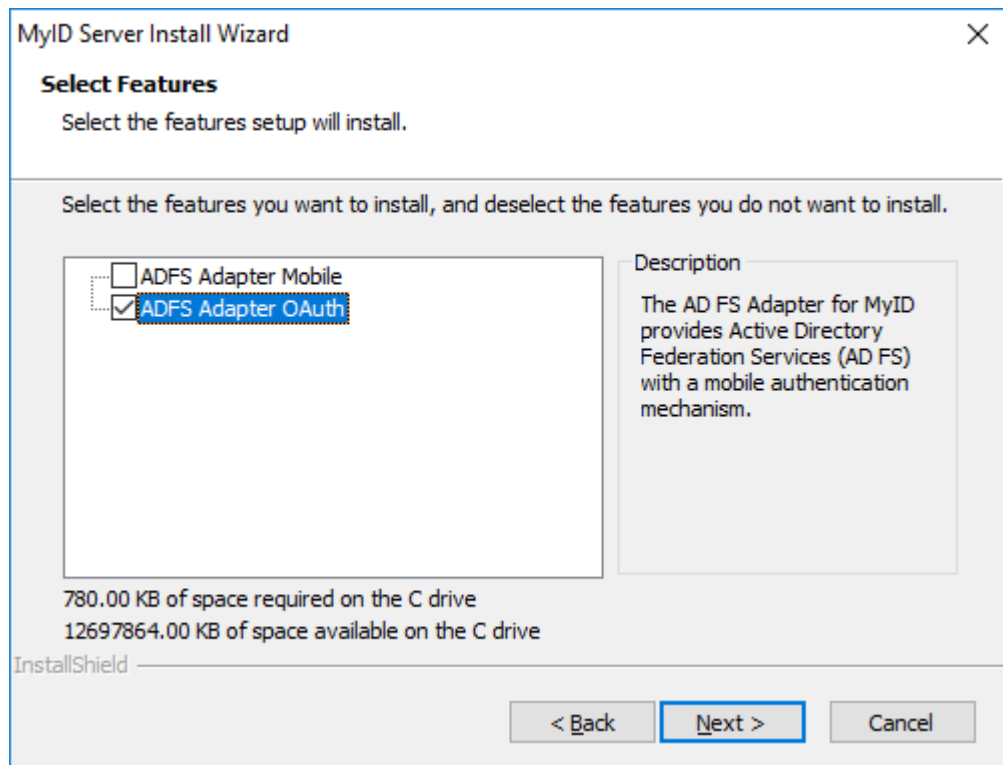
C:\Program Files\Intercede\

The installation program creates a the following folders in this location:

- ADFS_Adapter_OAuth – contains the AD FS Adapter configuration files.
- Themes – contains the themes for the AD FS Adapter;

Note: The themes folder is shared with the AD FS Adapter Mobile, if you have it installed.

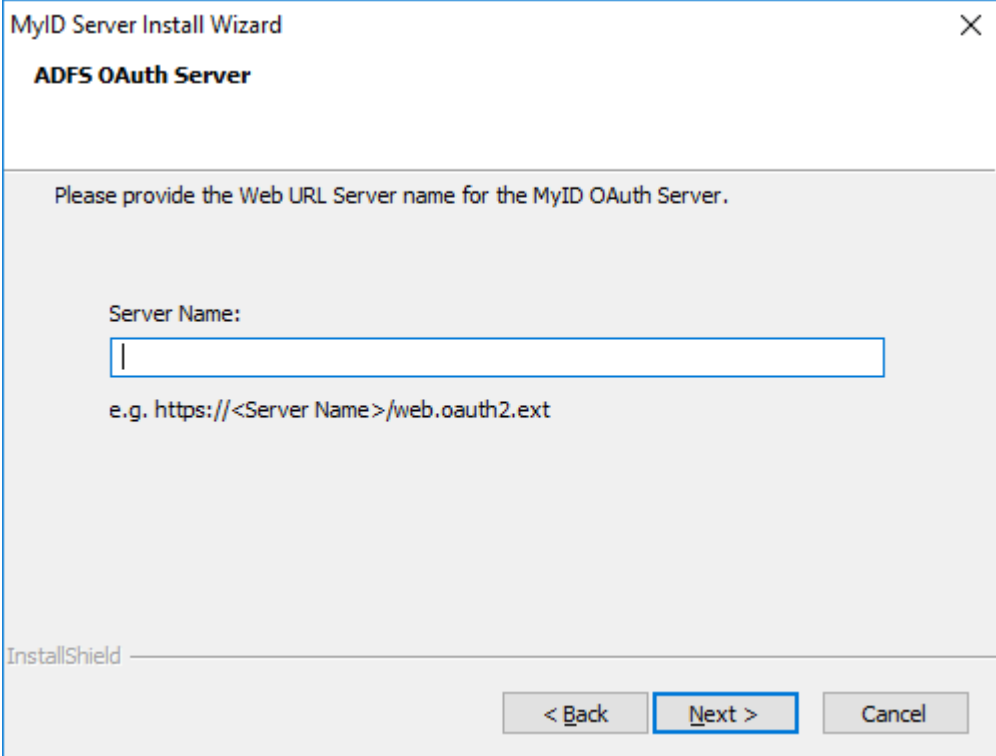
Click **Next**, and the Select Features screen appears.



5. Select the **ADFS Adapter OAuth** option.

For details of using the ADFS Adapter Mobile, see the *Installing the AD FS Adapter Mobile* section in the **Mobile Authentication** guide.

Click **Next**, and the ADFS OAuth Server screen appears.



6. Type the **Server Name** for your MyID standalone authentication server.

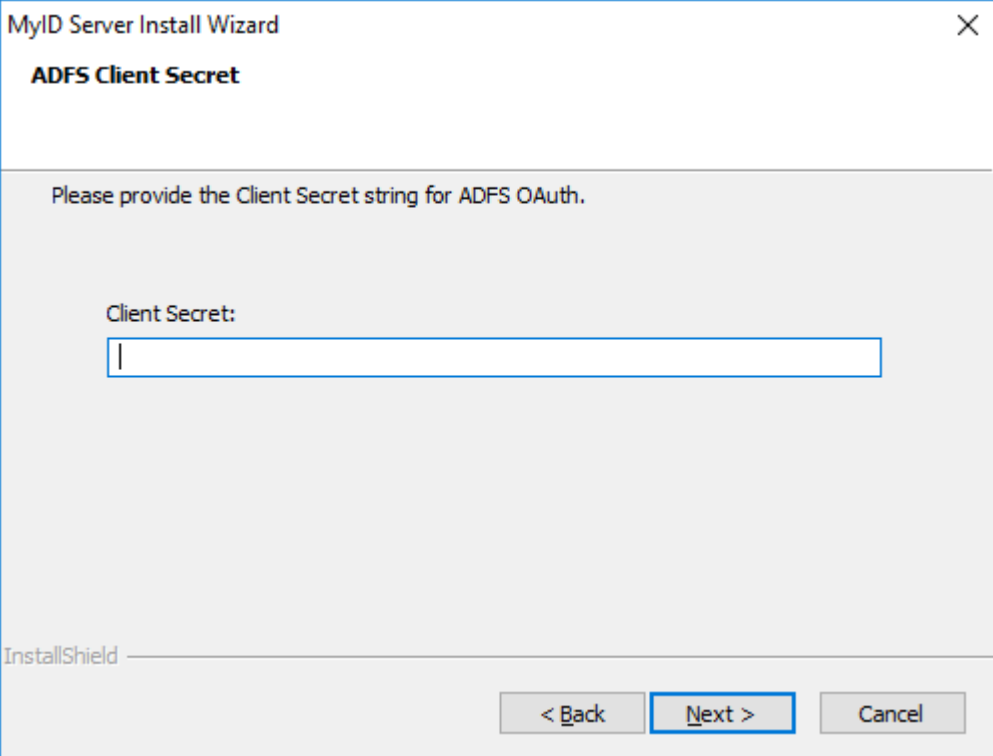
Note: Provide only the server name; do not include `https://` or the path to the `web.oauth2.ext` service. For example:

```
myserver.example.com
```

The installation program stores the server in the `Fido2AdfsAdapter.json` configuration file for both the standalone authentication service and the ADFS Auth web service, automatically completing the full URL; for example:

```
"myidAuth": {  
  "server": "https://myserver.example.com/web.oauth2.ext",  
  "redirect_server": "https://myserver.example.com/AdfsAuth"  
}
```

Click **Next**, and the ADFS Client Secret screen appears:



The screenshot shows a window titled "MyID Server Install Wizard" with a close button (X) in the top right corner. Below the title bar, the text "ADFS Client Secret" is displayed. The main content area contains the instruction "Please provide the Client Secret string for ADFS OAuth." followed by a label "Client Secret:" and a text input field. At the bottom left, the text "InstallShield" is visible. At the bottom right, there are three buttons: "< Back", "Next >" (which is highlighted with a blue border), and "Cancel".

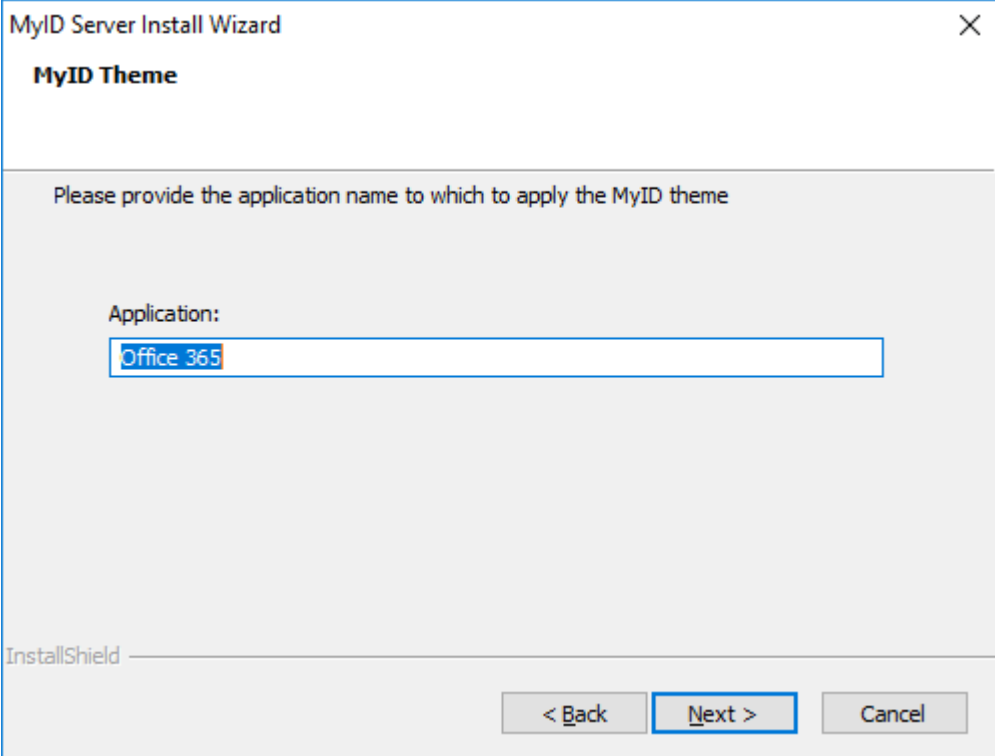
7. Provide the **Client Secret**.

See section [3.4.1, *Generating a shared secret*](#) for details of generating a shared secret and using it to secure the connection between the AD FS Adapter and the standalone authentication service.

The installation program stores this in the `Fido2AdfsAdapter.json` configuration file as an encrypted value:

```
"myidAuth": "client_secret"
```

Click **Next**, and the MyID Theme screen appears.



8. In the **Application** box, type the display name that was provided for the Relying Party Trust for which the AD FS Adapter OAuth will provide the authentication.

To find the display name, look in the following location:

Server Manager > Tools > AD FS Management > AD FS > Relying Party Trusts > Display Name

For more information on themes, see section [3.6.3, Managing themes](#).

Click **Next**, then click **Install**.

9. When the installation program has completed, click **Finish**.

3.5.1 Uninstalling the AD FS Adapter OAuth

You can uninstall the AD FS Adapter OAuth from the **Apps & features** section of Windows Settings; it is listed as the **AD FS Adapter for MyID**.

Note: Uninstalling the AD FS Adapter OAuth also uninstalls the AD FS Adapter Mobile, if you have it installed.

3.6 Managing the AD FS Adapter OAuth

After you have installed the AD FS Adapter OAuth, you can manage its settings using a provided suite of PowerShell scripts and a JSON configuration file.

3.6.1 Configuration file

The AD FS Adapter OAuth configuration is stored in a JSON file called `Fido2AdfsAdapter.json` in the `ADFS_Adapter_OAuth` folder; by default, this is:

```
C:\Program Files\Intercede\ADFS_Adapter_OAuth
```

If you have manually unregistered the AD FS Adapter OAuth and want to register it again, you can run the following PowerShell script :

- `RegisterADFSPProvider.ps1` – this script reads the information in the configuration file and uses it to register the AD FS Adapter OAuth.

You can also make changes to the configuration file and apply new settings.

To edit and apply new configuration settings:

1. In the `ADFS_Adapter_OAuth` folder, open the following file in a text editor:

```
Fido2AdfsAdapter.json
```

2. You can edit the following values:

- `server` – set this to the URL of the your standalone authentication server; for example:

```
https://myserver.example.com/web.oauth2.ext
```

- `client_secret` – set this to the client secret you set up to secure the connection between the AD FS Adapter OAuth and the standalone authentication server.

See section [3.4.1, *Generating a shared secret*](#) for details of creating the secret.

You can include the secret in plain text in the configuration file by setting the `client_secret_encrypted` option to `false`; however, you are strongly recommended to encrypt the secret. See section [3.6.2, *Encrypting the client secret*](#) for details.

- `redirect_server` – set this to the URL of the ADFS Auth web service; for example:

```
https://myserver.example.com/AdfsAuth
```

Do not change any of the other values in the configuration file.

3. Save the `Fido2AdfsAdapter.json` file.

4. Run the `ReconfigureADFSPProvider.ps1` PowerShell script to apply the changes.

This script unregisters the AD FS Adapter OAuth, then re-registers it using the updated settings.

3.6.2 Encrypting the client secret

You are strongly recommended to encrypt the client secret in the `Fido2AdfsAdapter.json` file. This is done automatically by the installation program, but if you need to update the client secret, you can encrypt it manually using the provided `DPAPIEncrypt.ps1` PowerShell script.

To encrypt the client secret:

1. Log on to the AD FS server using the account configured as the logon account for the AD FS service.

Note: It is important that you use this account to encrypt the secret, as no other accounts can decrypt the secret to use it.

2. Open a Windows PowerShell command prompt, and navigate to the `ADFS_Adapter_OAuth` folder.

By default, this is:

```
C:\Program Files\Intercede\ADFS_Adapter_OAuth
```

3. Run the following PowerShell script:

```
.\DPAPIEncrypt.ps1 <secret>
```

For example:

```
.\DPAPIEncrypt.ps1 82564d6e-c4a6-4f64-a6d4-cac43781c67c
```

The script outputs an encrypted copy of the secret; for example:

```
PS C:\Program Files\Intercede\ADFS_Adapter_OAuth> .\DPAPIEncrypt.ps1  
82564d6e-c4a6-4f64-a6d4-cac43781c67c  
AQAAANCMnd8BFdERjHoS [...] AAABGh5yPNcG7ubkY1aV93UrTxi7Daw==
```

(Encrypted output string truncated for documentation purposes.)

4. Copy the encrypted secret.
5. Edit the `Fido2AdfsAdapter.json` file, and set the following:
 - `client_secret` – set this to the encrypted secret.
 - `client_secret_encrypted` – set this to `true`.

For example:

```
"client_secret": "AQAAANCMnd8BFdERjHoS [...] AAABGh5yPNcG7ubkY1aV93UrTxi7Daw==",  
"client_secret_encrypted": "true",
```

6. Save the `Fido2AdfsAdapter.json` file.
7. Run the `ReconfigureADFSPProvider.ps1` PowerShell script to apply the changes.

This script unregisters the AD FS Adapter OAuth, then re-registers it using the updated settings.

3.6.3 Managing themes

After you have installed the AD FS Adapter, the Intercede branding files are stored in the `Themes` folder in the installation folder.

Note: The themes folder is shared between the AD FS Adapter OAuth and the AD FS Adapter Mobile, if you have both installed.

The `MyIDAuthTheme2019` folder contains files used for Windows Server 2019 or Windows Server 2022, and includes custom images, CSS, JavaScript and HTML. You are not expected to edit these files. The `MyIDAuthTheme` folder contains files previously used for systems running Windows Server 2016.

You can apply and remove these themes using the following PowerShell scripts:

- `ApplyCustomTheme.ps1`

This script applies the Intercede branding to the Relying Party Trust selected at installation time.

- `RemoveCustomTheme.ps1`

This script removes the Intercede branding from the Relying Party Trust selected at installation time.

3.6.4 Logging for the AD FS Adapter OAuth

Once you have installed and configured the AD FS Adapter OAuth as an authentication method for a Relying Party Trust, when an authentication starts it raises a Windows application event showing the configuration loaded when AD FS started the AD FS Adapter OAuth plug-in. This shows the latest AD FS Adapter OAuth configuration provided by the installer or reconfiguration script.

If the AD FS Adapter OAuth encounters a problem, it raises a Windows application error event describing the problem.

To see these events:

1. Open the Windows Event Viewer application.
2. Select **Windows Logs > Application**.

Events created in the Application event log by the AD FS Adapter OAuth have the source set to:

```
MyIDFidoAdfsAdapter
```

Additionally, if the calling AD FS service detects a problem from the AD FS Adapter OAuth, it raises an error event in the following location, describing the problem from the AD FS point of view:

Applications and Service Logs > ADFS > Admin

3.7 Troubleshooting

This section contains troubleshooting information and frequently asked questions related to working with the MyID AD FS Adapter OAuth.

- **I tried to log on, but I see "Incorrect user ID or password"**

The user or email address is incorrect. Correct it and try again.

- **I tried to log on, but I cannot see the FIDO option**

Check the ADFS Manager has selected the Intercede FIDO ADFS adapter as the Primary authentication method for Intranet.

- **I tried to log on, but I see "Your security cannot be used with this site"**

Try a different authenticator (security key). Some authenticators do not support user verification; that is, they do not have a PIN or fingerprint sensor.

- **I tried to log on, but I see "Error OA10010: Error authenticating FIDO in browser. The operation either timed out or was not allowed."**

This may be caused by the following situations:

- You exceeded the timeout (by default 90 seconds) before completing the authentication.

If necessary, you can change the timeout by adding the `Fido:Config:Timeout` option to the `appsettings.Production.json` file for the `web.oauth2.ext` web service.

- You canceled the authentication operation.

- **I tried to log on, but I see "400: An unexpected error occurred. Please contact your administrator."**

Check that the `Fido:Config:Origin` option in the `appsettings.Production.json` file is set correctly; see section [3.4.2, Updating the web.oauth2.ext configuration file](#) for details.

- **I tried to log on, but I see "Error OA10010: Error authenticating FIDO in browser. The relying party ID is not a registrable domain suffix of, nor equal to the current domain."**

Check that the `Fido:Config:ServerDomain` option in the `appsettings.json` or `appsettings.Production.json` file is set correctly; see the [Configuring the server settings](#) section in the [FIDO Authenticator Integration Guide](#) for details.

Ensure the domain name has not changed since the credential was issued – FIDO credentials can be used only for the domain on which they were registered.

If `web.oauth2.ext` (which is authenticating the user) is on a different server from `web.oauth2` (which registered the FIDO credential), ensure there is a load balancer or reverse proxy in front of both of these servers, so that the client computer sees the same domain in the URL for both of these machines.

- **I tried to log on, but I see "HTTP Error 500.30 - ANCM In-Process Start Failure"**

Check that your `appsettings.Production.json` file is valid.

Note especially that copying code samples from a browser may include hard spaces, which cause the JSON file to be invalid.

To assist in tracking down the problem, you can use the Windows Event Viewer. Check the **Windows Logs > Application** section for errors; you may find an error from the .NET Runtime source that contains information similar to:

```
Exception Info: System.FormatException: Could not parse the JSON file.
---> System.Text.Json.JsonReaderException: '"' is invalid after a
value. Expected either ',', '}', or ']'. LineNumber: 13 |
BytePositionInLine: 6.
```

which could be caused by a missing comma at the end of a line.

An error similar to:

```
Exception Info: System.FormatException: Could not parse the JSON file.
---> System.Text.Json.JsonReaderException: '0xC2' is an invalid start
of a property name. Expected a '"'. LineNumber: 7 | BytePositionInLine:
0.
```

is caused by a hard (non-breaking) space copied from a web browser, which is not supported in JSON.

Note: Some JSON files used by MyID contain comment lines beginning with double slashes `//` – these comments are not supported by the JSON format, so the JSON files will fail validation if you attempt to use external JSON validation tools. However, these comments *are* supported in the JSON implementation provided by `asp.net.core`, and so are valid in the context of MyID.

You can also check that the `Fido:Config:MDSAccessKeyClear` option in the `appsettings.Production.json` file is set correctly. If the `MDSAccessKey` contains an encrypted value, `MDSAccessKeyClear` must be `false`.

- **I tried to log on, but I see "This page isn't working <my domain> is currently unable to handle this request. HTTP ERROR 500"**

Check that the `MyID:Database:ConnectionStringCore` and `MyID:Database:ConnectionStringAuth` options in the `appsettings.json` or `appsettings.Production.json` file are set correctly.

See section 2.2, [Configuring the standalone authentication service](#) for details.

- **I tried to log on, but I see "Unknown error".**

- Check that the `myid.adfs` client in the `appsettings.Production.json` file has a valid value for `ClientSecrets`. This must be a Base64-encoded SHA-256 hash of the client secret.

See section [3.4, Configuring the standalone authentication service for AD FS](#).

- Check that the `Fido2AdfsAdapter.json` file on the AD FS server has a valid value for `client_secret`. This must be the shared secret that was used to generate the Base64-encoded SHA-256 hash that you used on the `web.oidc.ext` server; note that you are strongly recommended to use an encrypted secret in this file.

See section [3.6, Managing the AD FS Adapter OAuth](#) and section [3.6.2, Encrypting the client secret](#) for details.

- **I tried to log on, but I see "Sorry, there was an error : invalid_scope"**

Ensure that the `appsettings.json` file for the `web.oidc.ext` service has the following:

- In the `myid.adfs` client section, `AllowedScopes` of `openid` and `email`.
- An `IdentityResource` named `email`.

- **I tried to log on, but I see "Sorry, there was an error : invalid_request"**

Ensure that the `appsettings.Production.json` file for the `web.oidc.ext` service has in the `myid.adfs` client section a valid value for the `RedirectUri`.

This must be in the format:

```
https://<auth service domain>/AdfsAuth/home/AdfsAuth
```

See section [3.4.2, Updating the web.oidc.ext configuration file](#) for details.

- **I tried to log on, but I see "Invalid login request There are no login schemes configured for this client."**

Ensure that the `appsettings.Production.json` file for the `web.oidc.ext` service has in the `myid.adfs` client section, at least one of the following `Properties` set to `true`:

- `EnableFido2LoginBasicAssurance`
- `EnableFido2LoginHighAssurance`

See section [3.4.2, Updating the web.oidc.ext configuration file](#) for details.

- **I tried to log on, but I see "Error code OA10018: You do not have any FIDO tokens registered."**

Check the `Properties` for the `myid.adfs` client section in the `appsettings.Production.json` file for the `web.oidc.ext` service. This error can occur when `EnableFido2LoginBasicAssurance` is set to `true`, `EnableFido2LoginHighAssurance` is set to `false`, and while there may be authenticators registered with high assurance, there are no authenticators that were registered with basic assurance.

- **I tried to log on, but the screen stops responding with a message saying "Please wait a moment..."**

If this occurs, check the following:

- On the ADFS Auth web server, check that the `AllowedOrigins` option in the `appsettings.Production.json` file for the ADFS Auth web service is set correctly. It must be set to the URL of the AD FS server.

For example:

```
"AllowedOrigins": [ "https://adfs.example.com" ]
```

See section [3.3.2, Configuring the ADFS Auth web service](#) for details.

- On the AD FS server, check that the `FidoAdfsAdapter.json` file has a valid `redirect_server` URL setting.

For example:

```
https://myserver.example.com/AdfsAuth
```

See section [3.6.1, Configuration file](#) for details.

- **I tried to log on, but I see "Server Error, 404 – File or directory not found"**

On the AD FS server, check that the `FidoAdfsAdapter.json` file has a valid `server` URL setting.

For example:

```
https://myserver.example.com/web.oauth2.ext
```

See section [3.6.1, Configuration file](#) for details.

- **When the browser redirects to `web.oauth2.ext`, it says there are no logon mechanisms enabled**

Ensure that the `appsettings.Production.json` file for the `web.oauth2.ext` service has, in the `myid.adfs` client section, at least one of the following `Properties` set to `true`:

- `EnableFido2LoginBasicAssurance`
- `EnableFido2LoginHighAssurance`

See section [3.4.2, Updating the `web.oauth2.ext` configuration file](#) for details.

- **What are FIDO basic and high assurance?**

FIDO authenticators may provide single-factor, two-factor, or multi-factor authentication.; you can configure MyID to treat FIDO basic assurance authenticators and high assurance authenticators with different levels of trust; for example, you can enable logon to MyID for high assurance authenticators, but disable logon for basic assurance authenticators.

For more information about basic and high assurance FIDO authenticators, see the [Supported authenticators](#) section in the [FIDO Authenticator Integration Guide](#).

4 Authenticating using OpenID Connect

MyID provides a standards-based OAuth2 OpenID Connect authentication and authorization service that allows you to:

- Carry out server-to-server authentication for the MyID Core API.
See the *Server-to-server authentication* section in the [MyID Core API](#) guide.
- Carry out end-user authentication for the MyID Core API.
See the *End-user authentication* section in the [MyID Core API](#) guide.
- Carry out end-user authentication for your own external systems.
See section [4.1, Configuring the web service for OpenID Connect](#) and section [4.2, Obtaining an identity token](#).

The authentication service uses the following standards:

- OAuth2
This is the authorization framework used by the MyID authentication service.
For more information, see *The OAuth 2.0 Authorization Framework RFC*:
tools.ietf.org/html/rfc6749
- OpenID Connect
OpenID Connect is an identity layer on top of the OAuth 2.0 protocol that allows external systems to verify the identity of an end user based on the authentication performed by an authorization server (in this case, the MyID authentication service), as well as to obtain basic profile information about the end user.
For more information, see the OpenID Connect website:
openid.net/connect
- Proof Key for Code Exchange (PKCE)
PKCE is a system for securing requests for authorization codes and using them to request access or identity tokens.
For more information, see the *Proof Key for Code Exchange by OAuth Public Clients RFC*:
tools.ietf.org/html/rfc7636
- JSON Web Token (JWT)
JWT is a standard for the signed tokens that the MyID authentication service issues after authentication. This is either an access token (for example, used by a client to call the MyID Core API) or an identity token (for example, used by an external system to authenticate the identity of an end user).
For more information, see the *JSON Web Token (JWT) RFC*:
tools.ietf.org/html/rfc7519

- Fast IDentity Online (FIDO)

FIDO is a standard for interoperable authentication tokens.

For more information, see the FIDO Alliance website:

fidoalliance.org/fido2/fido2-web-authentication-webauthn

For information about integrating MyID with FIDO authenticators, see the ***FIDO Authenticator Integration Guide***.

4.1 Configuring the web service for OpenID Connect

You can use either the standard authentication service (`web.oauth2`) or the standalone authentication service (`web.oauth2.ext`) to authenticate your users. The standalone authentication service does not require access to the MyID application server, but connects directly to the database; however, currently the standalone authentication service supports only FIDO as an authentication method.

For more information on the standalone authentication service, see section 2, [Setting up the standalone authentication service](#).

Before you can use the MyID authentication service to verify the identity of an end user, you must configure the web service with the details of your external system and with the security protocols you want to use.

You can secure the request for an identity token in the following ways:

- Using a shared secret.

When configuring your authentication service, you generate a secret code, then store a Base64-encoded SHA-256 hash of the code on the server; when you make the authentication code request, you provide your secret code securely over https, and the server compares it to the hash. The secret code is never stored on the server.

Important: You must keep the shared secret safe, as it can provide access to the features of the MyID authentication service. Do not use the same shared secret for end-user authentication and server-to-server API authentication.

- Using PKCE.

For each authentication code request, you generate a secret code (known as the *code verifier*), then pass a Base64 URL-encoded SHA-256 hash of the code (known as the *code challenge*) in the request. When you exchange the authentication code for an identity token, you provide the code verifier, which the server compares to the code challenge to verify that the same system requested the original authentication code.

For more information on PKCE, including details of requirements for the code verifier and code challenge, see the *Proof Key for Code Exchange by OAuth Public Clients* RFC:

tools.ietf.org/html/rfc7636

You can use one or both of these methods. The examples in this document assume that you are using a combination of both methods.

For stateful websites, where for example the server uses cookies to map stateful sessions between the client and the web server, it is recommended to configure the authentication service to require a client secret; you do not have to use PKCE, but you can use it in addition to the client secret if you want.

For single-page apps, which run entirely on the client PC, you must secure the request for authentication using PKCE; a shared secret is not appropriate.

Note: You must use TLS to ensure the security of the system. The MyID authentication service is configured to use TLS by default; you must not use OAuth2 OpenID Connect-based systems without TLS.

4.1.1 Creating a shared secret

If you are using a shared secret, you must generate an unguessable secret and create a hash to store on the server.

You are recommended to use a GUID for the secret.

If you are using the standalone authentication web service (web.oauth2.ext) you can use the provided `GenClientSecret.ps1` PowerShell script to generate a GUID and create a SHA-256 hash converted to Base64:

1. On the web server, open a Windows PowerShell command prompt.
2. Navigate to the authentication service folder.

```
C:\Program Files\Intercede\MyID\web.oauth2.ext
```

3. Run the script:

```
.\GenClientSecret.ps1
```

The script generates a GUID to use as the shared secret, creates a SHA-256 hash from the GUID, then converts the hash to Base64; for example:

```
PS C:\Program Files\Intercede\MyID\web.oauth2> .\GenClientSecret.ps1
client secret: 82564d6e-c4a6-4f64-a6d4-cac43781c67c
SHA256+base64: kv3lVP5z/oKS0QMMaIfZ2UrhmqOdgAPpXV/vaFlcymk=
```

Important: Do not use this example secret in your own system.

4. Take a note of the following values:
 - `client secret` – you need this value for the `client_secret` parameter when posting to the token URL.
See section [4.2.3, Requesting an identity token](#).
 - `SHA256+base64` – you need this value for the `ClientSecrets` parameter in the web service configuration file. The server does not store the secret, only the Base64-encoded hash.
See section [4.1.2, Editing the configuration file](#).

The script uses the following code to generate the GUID and Base64-encoded hash:

```
Add-Type -AssemblyName System.Security
if ($args.count -ne 0)
{
    write-host "Usage: GenClientSecret"
    exit
}
$clientSecret = [guid]::NewGuid()

write-host "client secret: " $ClientSecret

$hasher = [System.Security.Cryptography.HashAlgorithm]::Create('sha256')
$hashBytes = $hasher.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($ClientSecret))

$Sha256Base64 = [Convert]::ToBase64String($hashBytes)
write-host "SHA256+base64: " $Sha256Base64
```

4.1.2 Editing the configuration file

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2\appsettings.Production.json
```

If you are using a standalone web service, by default, this is:

```
C:\Program  
Files\Intercede\MyID\web.oauth2.ext\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service.

2. Edit the file to include the following:

```
{
  "Clients": [
    {
      "ClientId": "<my client ID>",
      "ClientName": "<my client name>",
      "AccessTokenLifetime": <access lifetime>,
      "IdentityTokenLifetime": <identity lifetime>,
      "AllowedGrantTypes": [
        "authorization_code"
      ],
      "RequireClientSecret": true,
      "RequirePkce": true,
      "AllowAccessTokensViaBrowser": true,
      "RequireConsent": true,
      "ClientSecrets": [
        {
          "Value": "<secret>"
        }
      ],
      "AllowedScopes": [
        "openid"
      ],
      "AlwaysIncludeUserClaimsInIdToken": true,
      "RedirectUris": [
        "<callback URL>"
      ],
      "AllowedCorsOrigins": [
        "<origin>"
      ],
      "Properties": {
        "EnableSelfService": true,
        "Skin": "popover"
      }
    }
  ]
}
```

where:

- <my client ID> – the client ID you decided on; for example:
myid.openid
This represents your back-end system that intends to make calls to the MyID authentication service.
- <my client name> – an easily readable name for your client system; for example:
My OpenID Connect System
- <access lifetime> – the time (in seconds) that the access token is valid. The default is 3600 – 1 hour.
- <identity lifetime> – the time (in seconds) that the identity token is valid. The default is 300 – 5 minutes.
- <secret> – if you are using a shared secret, set this to the Base64-encoded SHA-256 hash of the secret you created; for example:

kv31VP5z/oKS0QMMaIfZ2UrhmqOdgAppXV/vaF1cymk=

See section [4.1.1, Creating a shared secret](#).

- `<callback URL>` – the URL of the web page on your system to which the authorization code will be returned.
- `<origin>` – used for Cross-Origin Resource Sharing (CORS). If the web page that calls the authentication service is on a different origin from the authentication service, you must add the origin to this list.

Note: Make sure you use an origin, and not an URL, when configuring CORS. For example: `https://myserver/` is an URL, while `https://myserver` is an origin.

You can set the following options:

- `RequireClientSecret` – set this to `true` if you are using a shared secret to secure the request.
- `RequirePkce` – set this to `true` if you are using PKCE to secure the request.
Note: You must use a shared secret, PKCE, or both a shared secret *and* PKCE.
- `RequireConsent` – set this to `true` to allow the end-user to approve the external system's access to their identity information stored on your authentication service.
- `AllowedScopes` – set this to one of the values in the `IdentityResources` section of the `appsettings.json` file for the authentication service:
 - `openid` – returns the `sub` claim in the identity token; this is the subject ID of the end user, as stored in the `UserAccounts.ObjectID` field in the MyID database.
 - `email` – returns the `email` claim in the identity token; this is the email address of the user, as stored in the `People.Email` field in the MyID database.
Note: The email claim is currently available only for FIDO authentication.
 - `profile` – returns the `name` claim in the identity token; this is the MyID logon name of the user, as stored in the `UserAccounts.LogonName` field in the MyID database.
- `AlwaysIncludeUserClaimsInIdToken` – set this to `true` to allow claims other than `sub` to be returned in the identity token. If you do not set this to `true`, setting `AllowedScopes` to include `email` and `profile` does not result in additional claims in the identity token.
- `Properties` – you can set the following:
 - `Skin` – currently only `popover` is available as a skin option for the authentication GUI. The authentication web page shows a background image with a popover representing the authentication UI; this is suitable for large windows. If you are displaying the authentication page in a small popup window, you can omit this option.
 - `EnableFido2LoginBasicAssurance` – set this to `true` to allow authentication using FIDO basic assurance authenticators, which provide only single-factor authentication.
 - `EnableFido2LoginHighAssurance` – set this to `true` to allow authentication using FIDO high assurance authenticators, which provide multi-factor authentication.

When using FIDO with `web.oidc2`, authentication using high and basic assurance is controlled using the logon mechanisms feature; as the `web.oidc2.ext` standalone authentication service does not use the MyID application server, this feature is not available, and you must specify the logon mechanisms using the above options.

For more information about basic and high assurance FIDO authenticators, see the *Supported authenticators* section in the [FIDO Authenticator Integration Guide](#).

- `EnableAuthCodeLogin` – set this to `true` to allow authentication using single-use authentication codes (assuming MyID is configured to use the **Authentication Code** logon mechanism), or `false` to prevent this client from using single-use authentication codes, even if MyID is configured for authentication codes for the MyID Operator Client.

For more information on authentication codes, see the *Configuring authentication codes for the MyID authentication server* section in the [Administration Guide](#).

- `EnablePassphraseLogin` – set this to `true` to allow authentication using security phrases (assuming MyID is configured to use the **Password Logon** logon mechanism) or `false` to prevent this client from using security phrases, even if MyID is configured for security phrase logon for the MyID Operator Client.

The default is `true`, which allows security phrase logon if MyID is configured to use **Password Logon** in the **Logon Mechanisms** tab of the **Security Settings** workflow.

- `EnableCardLogin` – set this to `true` to allow authentication using a smart card (assuming MyID is configured to use the **Smart Card Logon** logon mechanism) or `false` to prevent this client from using smart cards to authenticate, even if MyID is configured for smart card logon for the MyID Operator Client.

The default is `true`, which allows smart card logon if MyID is configured to use **Smart Card Logon** in the **Logon Mechanisms** tab of the **Security Settings** workflow.

- `EnableWindowsLogin` – set this to `true` to allow authentication using a Integrated Windows Logon (assuming MyID is configured to use the **Integrated Windows Logon** logon mechanism) or `false` to prevent this client from using Windows credentials to authenticate, even if MyID is configured for Integrated Windows Logon for the MyID Operator Client.

The default is `true`, which allows Integrated Windows Logon if MyID is configured to use **Integrated Windows Logon** in the **Logon Mechanisms** tab of the **Security Settings** workflow.

See the *Signing in using Windows authentication* section in the [MyID Operator Client](#) guide for more information about configuring MyID for Integrated Windows Logon.

- `EnableHeadlessCardLogin` – set this to `true` to allow authentication using a key pair; for example, for certificate renewal for mobile identities. MyID must also be configured to use the **Smart Card Logon** logon mechanism.

The default is `false`. This setting is currently used only for the `myid.rest.mobile` client.

- `EnableHeadlessPassphraseLogin` – set this to `true` to allow authentication using a key pair; for example, for certificate renewal for mobile identities. MyID must also be configured to use the **Password Logon** logon mechanism.

The default is `false`. This setting is currently used only for the `myid.rest.mobile` client.

- `EnableSelfService` – set this to `true` to allow people to launch the MyID Self-Service App from the MyID Authentication screen using the **Manage My Credentials** link; this allows people to manage their credentials without first signing in to MyID; for example, to change their security phrases or reset their device PIN. Set this property to `false` to prevent the **Manage My Credentials** link from being displayed.

Note: The **Allow Self-Service at Logon** configuration option (on the **Logon** tab of the **Security Settings** workflow) must also be set for this option to appear.

- `EnableLoginMechanism_<id>` – where `<id>` is the ID of the external identity provider logon mechanism; for example, `EnableLogonMechanism_121`. Set this to `true` to allow authentication using the specified logon mechanism (assuming MyID is configured to use the appropriate logon mechanism in the **Logon Mechanisms** tab of the **Security Settings** workflow).

See section [6](#), *Setting up an external identity provider* for more information.

Important: If you have clients in the `appsettings.json` file *and* the `appsettings.Production.json` file, make sure the production file does not overwrite the entries in the base file. In these settings files, entries in arrays are determined by their index; therefore if you have four existing entries in the `appsettings.json` file, you must include four blank array entries `{}`, in the `appsettings.Production.json` file before you include your new client details. Alternatively, you can include the entire `Clients` array in the `appsettings.Production.json` file.

Note: By default, the `appsettings.Production.json` file contains the `myid.adfs` client section in the appropriate place; see section 3.4, [Configuring the standalone authentication service for AD FS](#) for details.

For example:

```
{
  "Clients": [
    {},
    {},
    {},
    {},
    {
      "ClientId": "myid.openid",
      "ClientName": "My OpenID Connect System",
      "AccessTokenLifetime": 3600,
      "IdentityTokenLifetime": 300,
      "AllowedGrantTypes": [
        "authorization_code"
      ],
      "RequireClientSecret": true,
      "RequirePkce": true,
      "AllowAccessTokensViaBrowser": true,
      "RequireConsent": true,
      "ClientSecrets": [
        {
          "Value":
"kv31VP5z/oKS0QMMaIfZ2Urhmq0dgAppXV/vaF1cymk="
        }
      ],
      "AllowedScopes": [
        "openid"
      ],
      "AlwaysIncludeUserClaimsInIdToken": true,
      "RedirectUris": [
        "https://myserver/mysystem/callback.asp"
      ],
      "AllowedCorsOrigins": [
        "http://myserver"
      ],
      "Properties": {
        "EnableSelfService": true,
        "Skin": "popover"
      }
    }
  ]
}
```

If you already have an `appsettings.Production.json` file, back up the existing file, then incorporate the new client section above into the file.

3. Save the configuration file.
4. Recycle the web service app pool:
 - a. On the MyID web server, in Internet Information Services (IIS) Manager, select **Application Pools**.
 - b. Right-click the **myid.web.oauth2.pool** application pool, then from the pop-up menu click **Recycle**.
 - c. If you are using the standalone web service, right-click the **myid.web.oauth2.ext.pool** application pool, then from the pop-up menu click **Recycle**.

This ensures that the web service has picked up the changes to the configuration file.

5. Check that the authentication service is still operational by logging on to the MyID Operator Client.

Application setting JSON files are sensitive to such issues as comma placement; if the format of the file is not correct, the web service cannot load the file and will not operate, which may result in an error similar to:

```
HTTP Error 500.30 - ANCM In-Process Start Failure
```

See section [4.3, *Troubleshooting*](#) for information on resolving problems that cause HTTP Error 500.30.

As an alternative to logging on to the MyID Operator Client (for example, if you are using the standalone authentication service), you can check the following URL:

```
https://<myserver>/web.oauth2/.well-known/openid-configuration
```

For the standalone authentication service, this is:

```
https://<myserver>/web.oauth2.ext/.well-known/openid-configuration
```

where `<myserver>` is the address of the MyID authentication service. This should return a block of JSON describing the endpoints and configuration of your authentication service.

4.1.3 Configuring authentication to skip the MyID Authentication screen

You can configure authentication for web.oauth2 or the MyID Operator Client to skip the MyID Authentication screen; if there is a single logon mechanism enabled (due to other logon mechanisms being disabled in MyID security settings or in the web.oauth2 appsettings for a given oauth2 client) the user is not presented with the MyID Authentication screen when they authenticate. The single available logon mechanism that is enabled is automatically used. However, if the `EnableSelfService` option is set to `true` the MyID Authentication screen also displays a **Manage My Credentials** link; this means that MyID cannot skip the MyID Authentication screen.

4.2 Obtaining an identity token

Obtaining an identity token is a two-stage process. First, you must request an authorization code; once you have this code, you can exchange it for an identity token.

4.2.1 Generating a PKCE code verifier and code challenge

If you are using PKCE, you must generate a code verifier and code challenge. each time you request an authorization code and identity token.

The PKCE code verifier and code challenge are used to request the authorization code.

1. Generate a cryptographically-random key.

This is the *code verifier*.

The code verifier must be a high-entropy cryptographic random string using the following characters:

```
[A-Z] / [a-z] / [0-9] / "-" / "." / "_" / "~"
```

The minimum length is 43 characters, and the maximum length is 128 characters.

2. Generate a SHA-256 hash of this key, then encode it using Base64 URL encoding.

This is the *code challenge*.

Important: Base64 URL encoding is slightly different to standard Base64 encoding.

See the *Protocol* section of the PKCE standard for details of requirements for the code verifier and code challenge:

tools.ietf.org/html/rfc7636#section-4

Example PowerShell script for generating a code challenge from a given code verifier:

```
$code_verifier = 'TiGVEDHIRkdTpif4zLw8v6tcdG2VJXvp4r0fuLhsXIj'

# Hash the code verifier using SHA-256
$hasher = [System.Security.Cryptography.HashAlgorithm]::Create("sha256")
$hashOfSecret = $hasher.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($code_verifier))

# Convert to Base64 URL encoded (slightly different to normal Base64)
$clientSecret = [System.Convert]::ToBase64String($hashOfSecret)
$clientSecret = $clientSecret.Split('=')[0]
$clientSecret = $clientSecret.Replace('+', '-')
$clientSecret = $clientSecret.Replace('/', '_')

# Output the results
Write-Output "`n`nThe code verifier is: `n`n`n$code_verifier"
Write-Output "`n`nAnd code challenge is:`n`n`n$clientSecret" )

# Wait for a keypress
Write-Host "`n`nPress any key to continue...`n`n" -ForegroundColor Yellow
[void][System.Console]::ReadKey($true)
```

4.2.2 Requesting an authorization code

You must request an authorization code from the authentication service before you can obtain an identity token.

1. From your website, post the following information to the MyID authorization URL:

`https://<server>/web.oauth2/connect/authorize`

For the standalone authentication service, this is:

`https://<server>/web.oauth2.ext/connect/authorize`

- `client_id` – the ID of your system; for example:
`myid.openid`
- `scope` – set this to one of the scopes listed in the `AllowedScopes` parameter of your `appsettings.Production.json` file; for example:
`openid`
- `redirect_uri` – set this to the URL of the page on your website to which the authorization code will be returned. This must be the same as the URL you specified in the `appsettings.Production.json` file.
- `response_type` – set this to `code`
- `code_challenge` – if you are using PKCE, set this to the PKCE code challenge you generated. This is the Base64 URL-encoded SHA-256 hash of the random code verifier you created.
- `code_challenge_method` – if you are using PKCE, set this to `S256`
- `state` – this value is returned in the redirect, allowing you to persist data between the authorization request and the response; you can use this as a session key.

- `acr_values` – an optional parameter that allows you to pass a space-separated list of values. Currently, this supports:
 - `logonmechanism:<logon_mechanism>` – allows you to select which logon mechanism to use. Use the following values:

Value	Logon Mechanism	appsettings
passphrase	Password Logon	EnablePassphraseLogin
smartcardms	Smart Card Logon	EnableCardLogin
fido	FIDO Basic Assurance FIDO High Assurance	EnableFido2LoginBasicAssurance EnableFido2LoginHighAssurance
authcode	Authentication Code	EnableAuthCodeLogin
windows	Windows Authentication	EnableWindowsLogin
<id>	External identity provider	EnableLoginMechanism_<id>

The relevant authentication mechanism must be enabled in the **Logon Mechanisms** tab of the **Security Settings** workflow in MyID, and must not be overridden with a `false` value in the `Properties` section for the selected client in the `appsettings` file.

For example:

- `acr_values=logonmechanism:authcode`
Specifies the Authentication Code logon mechanism.
- `acr_values=logonmechanism:101`
Specifies the Microsoft Entra external identity provider (which has the external identity provider logon mechanism ID of 101).

If you provide this setting in an URL, you must URL-encode the `:` in the parameter: use `%3a` as a substitute; for example:

```
acr_values=logonmechanism%3aauthcode
acr_values=logonmechanism%3a101
```

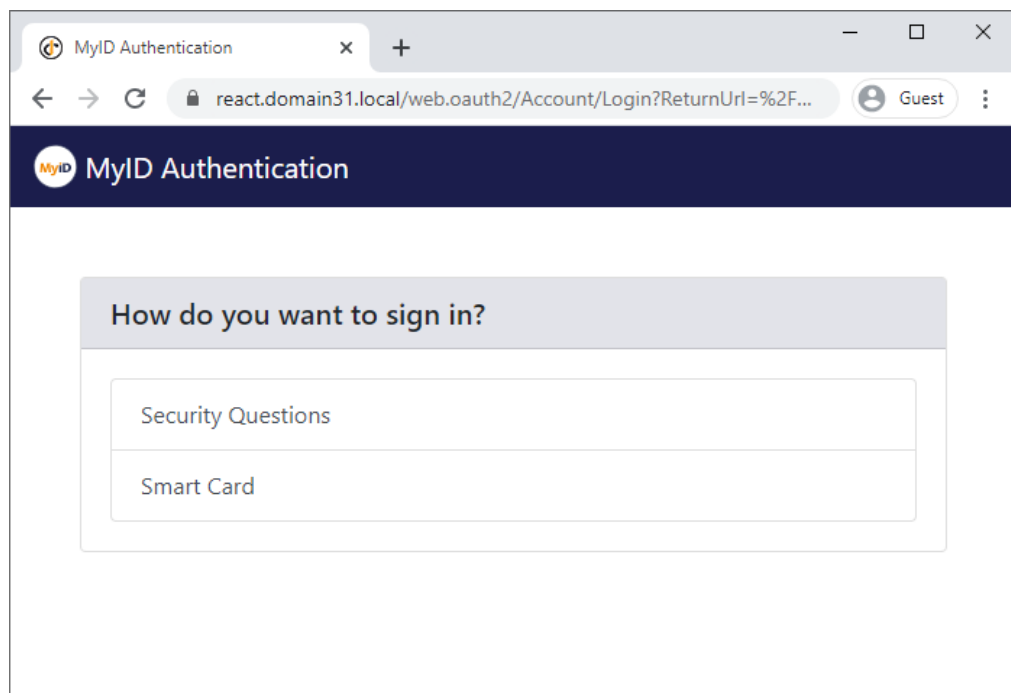
- `login_hint` – an optional parameter that allows you to provide the logon user name when using the Authentication Code logon mechanism. The **Username or email address** field on the Authentication Code Login dialog is pre-populated with the value you provide here.

You must URL-encode the logon user name; use `%20` as a substitute for a space. For example:

```
login_hint=susan%20smith
```

When you post this request, the MyID authentication service prompts you for your user credentials. The available methods of authentication depend on how you have configured your system; the same methods are available for authentication as are available in the MyID Operator Client. (Although note that you can disable logon mechanisms for each client listed in the `web.oauth2` application settings file; see section [4.1.2, Editing the configuration file.](#))

If you are using the standalone authentication service, only FIDO is currently supported as a logon method.



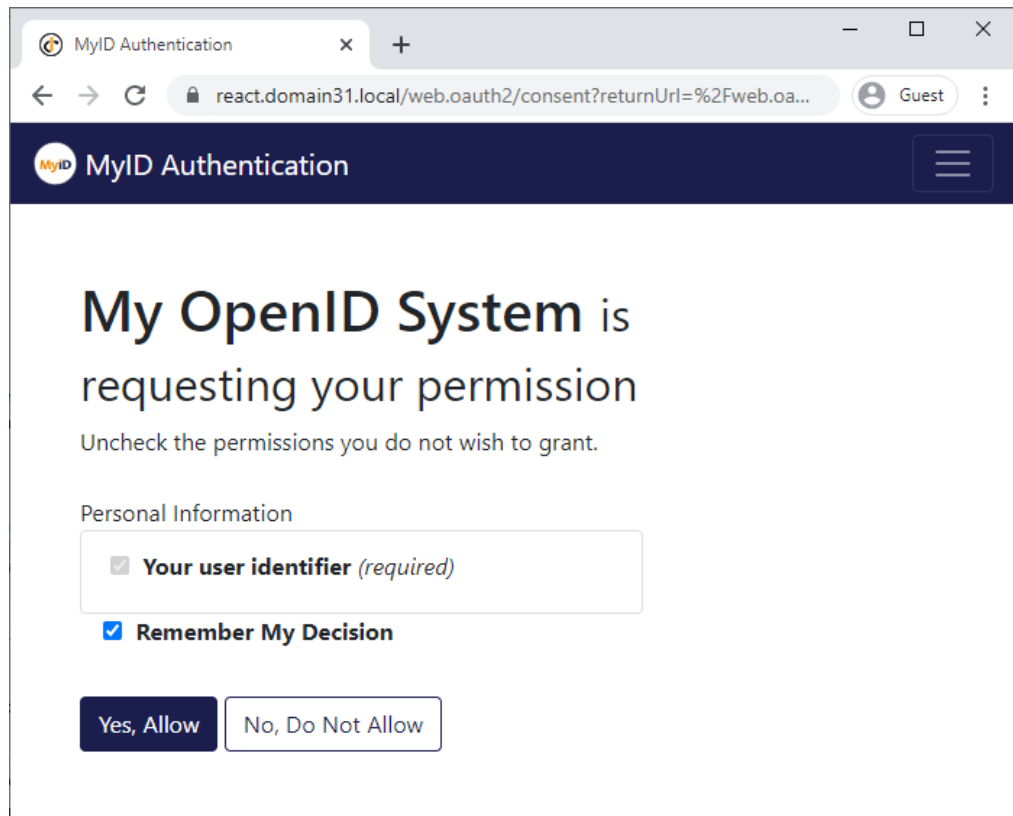
If the `EnableSelfService` option is set to `true` (see section [4.1.2, Editing the configuration file](#)) the MyID Authentication screen also displays a **Manage My Credentials** link that allows you to launch the MyID Self-Service App to manage your credentials (for example, changing your security phrases or resetting your device PIN) without completing your authentication. You must have the MyID Self-Service App installed, and the MyID Client Service app installed and running, to use this feature.

Note: If there is only one authentication method available, this stage is skipped. This also means that you are not given an opportunity to use the **Manage My Credentials** link.

2. Complete the authentication using your method of choice; for example, security questions or FIDO.

Note: You may need to ensure that the MyID Client Service is running on your PC; for example, smart cards and VSCs require the MyID Client Service.

If you configured your system to require consent using the `RequireConsent` parameter, the end user must allow your external system to access their information on the MyID authentication service:



Note: The name displayed on this screen is the `ClientName` parameter you set up in the `appsettings.Production.json` configuration file.

3. Capture the `code` parameter that is returned to the page you specified in the `redirect_uri` parameter.

This is your authorization code, which can be used once to request an identity token. If you need to request another identity token, you must first request another authorization code and go through the user authentication procedure again.

4.2.3 Requesting an identity token

Once you have an authorization code, you can exchange it for an identity token.

1. Post the following information to the MyID token URL:

```
https://<server>/web.oauth2/connect/token
```

For the standalone authentication service, this is:

```
https://<server>/web.oauth2.ext/connect/token
```

- `grant_type` – set this to `authorization_code`
- `client_id` – the ID of your system; for example:
`myid.openid`

This must be the same as the client ID you used to request the authorization code.

- `code_verifier` – if you are using PKCE, set this to the code verifier you created.
Note: Do not use the Base64 URL-encoded SHA-256 hash (the code challenge) – use the original plaintext value. The server compares this value to the encoded hash you provided when you requested the authorization code.
- `client_secret` – if you are using a shared secret, set this to the plaintext of the client secret you configured for the authentication service.
Note: Alternatively, you can combine the `client_id` and the `client_secret` and post them as a Basic authentication header; for an example of this, see the *Requesting an access token* section in the *MyID Core API* document.
- `code` – set this to the authorization code you obtained from the server.
- `redirect_uri` – set this to the URL of the page on your website you specified when you requested the authorization code.

2. Capture the `id_token` that is returned.

The identity token is in the standard JWT format. You can inspect the token manually at jwt.io or use a variety of existing libraries or middleware to extract the identity information from the token.

For example, the `openid` scope returns the `sub` claim in the payload of the identity token:

```
"sub": "5d3eac85-fa64-4891-b98a-52412b0c585d"
```

This is the subject ID of the end user, as stored in the `UserAccounts.ObjectID` field in the MyID database.

See the information about the `AllowedScopes` parameter in section [4.1.2, Editing the configuration file](#) for details about other claims that you can request.

Note: You must set the `AlwaysIncludeUserClaimsInIdToken` parameter to `true` to allow claims other than `sub` to be returned in the identity token.

4.3 Troubleshooting

This section contains troubleshooting information for OpenID authentication.

If you are experiencing problems, you are recommended to enable logging for the `web.oauth2` or `web.oauth2.ext` web service; see the *MyID REST and authentication web services* section in the [Configuring Logging](#) guide for details.

To confirm that the authentication service is running, you can check the following URL:

```
https://<myserver>/web.oauth2/.well-known/openid-configuration
```

For the standalone authentication service, this is:

```
https://<myserver>/web.oauth2.ext/.well-known/openid-configuration
```

where `<myserver>` is the address of the MyID authentication service. This should return a block of JSON describing the endpoints and configuration of your authentication service.

- **HTTP Error 500.30 when accessing the authentication service**

If you see an error similar to:

```
HTTP Error 500.30 - ANCM In-Process Start Failure
```

Check that your `appsettings.Production.json` file is valid.

Note especially that copying code samples from a browser may include hard spaces, which cause the JSON file to be invalid.

To assist in tracking down the problem, you can use the Windows Event Viewer. Check the **Windows Logs > Application** section for errors; you may find an error from the .NET Runtime source that contains information similar to:

```
Exception Info: System.FormatException: Could not parse the JSON file.
---> System.Text.Json.JsonReaderException: '"' is invalid after a
value. Expected either ',', '}', or ']'. LineNumber: 13 |
BytePositionInLine: 6.
```

which could be caused by a missing comma at the end of a line.

An error similar to:

```
Exception Info: System.FormatException: Could not parse the JSON file.
---> System.Text.Json.JsonReaderException: '0xC2' is an invalid start
of a property name. Expected a '"'. LineNumber: 7 | BytePositionInLine:
0.
```

is caused by a hard (non-breaking) space copied from a web browser, which is not supported in JSON.

Note: Some JSON files used by MyID contain comment lines beginning with double slashes `//` – these comments are not supported by the JSON format, so the JSON files will fail validation if you attempt to use external JSON validation tools. However, these comments *are* supported in the JSON implementation provided by `asp.net.core`, and so are valid in the context of MyID.

- **invalid_grant error when requesting an identity token**

This is a general error that may have several causes. Check the log for the web.oauth2 or web.oauth2.ext service for more information.

For example, you may see errors in the log similar to:

- `code_verifier is too short or too long`

In this case, check that your PKCE code verifier is between 43 and 128 characters. Note that this means a GUID is not long enough.

- `Client is trying to use a code from a different client`

In this case, check that the `client_id` you passed when requesting the identity token is the same as the `client_id` you passed when requesting the authorization code.

- `Invalid redirect_uri`

In this case, the redirect URL that you provided in the request for an identity token does not match the URL that you provided when requesting the authorization code; you must use the same URL in both requests.

- `Invalid authorization code`

In this case, you have either provided an incorrect authorization code, or the code has expired. Authorization codes are single-use; even if the attempt to obtain an identity token fails for another reason, you cannot re-use an authorization code. You must request another code and start the process again.

- **invalid_scope error when requesting an authorization code**

The scope you requested is not permitted for the client ID you are using. Check that you have set up the `AllowedScopes` parameter correctly.

Check the log for the web.oauth2 or web.oauth2.ext service for more information.

- **unauthorized_client error when requesting an authorization code**

Check that the client ID you are passing is correct; this must be a valid `ClientId` from the `appsettings.Production.json` file.

Check the log for the web.oauth2 or web.oauth2.ext service for more information.

- **unsupported_response_type error when requesting an authorization code**

Check that the response type you are requesting is `code`.

Check the log for the web.oauth2 or web.oauth2.ext service for more information.

- **Cannot access the MyID Operator Client**

If your new system is working correctly, but when you attempt to sign in to the MyID Operator Client you see an error similar to:

```
Sorry, there was an error : unauthorized_client
```

You may have overwritten the standard clients in the `appsettings.json` file with your changes to the `appsettings.Production.json` file.

The log for the `web.oidc` or `web.oidc.ext` service will contain an error similar to:

```
Unknown client or not enabled: myid.operatorclient
```

In JSON settings files, entries in arrays are determined by their index; therefore if you have four existing entries in the `appsettings.json` file, you must include four blank array entries `{}`, in the `appsettings.Production.json` file before you include your new client details.

5 Authenticating for embedded Operator Client screens

You can embed MyID Operator Client screens in your own intranet pages. Each screen has a dedicated URL that you can use to specify the page; for example, the Add Person screen, or a list of search results, or the View Person screen for a particular person.

See the *Using the browser location bar* section in the *MyID Operator Client* guide for details of Operator Client URLs.

You can embed the screen in an iframe, authenticate to MyID, then carry out any required actions.

However, if you intend to view more than one screen, you may not want to have to authenticate to the MyID Operator Client for each screen. Instead, you can configure the MyID authentication server to allow you to request an authentication token, which you can then post to the iframe containing the MyID Operator Client screen.

The process is as follows:

1. Configure the authentication server.
See section 5.1, *Configuring web.oidc for user-based authentication*.
2. Authenticate to MyID and obtain an access token.
See section 5.2, *Obtaining an access token*.
3. Post the access token to the embedded Operator Client screen.
See section 5.3, *Posting the access token*.

5.1 Configuring web.oidc for user-based authentication

The *MyID Core API* guide contains details of configuring the web.oidc authentication server for user-based authentication using PKCE. Follow the instructions in the *Configuring the authentication service for PKCE* section, setting the following:

- `ClientId` – set this to an ID for your intranet; for example:
`myid.intranet`
- `RedirectUris` – add the location of your intranet page to which the authorization code will be returned.

For example:

```
"RedirectUris": [  
  "https://react.domain31.local/mysystem/callback.asp"  
]
```

5.2 Obtaining an access token

Follow the instructions in *Obtaining an end-user based access token using PKCE* section of the **MyID Core API** guide. You must carry out the following:

1. Generate a PKCE code verifier and code challenge.
See the *Generating a PKCE code verifier and code challenge* section.
2. Obtain an authorization code from the authentication server, passing the PKCE code challenge.

See the *Requesting an authorization code* section.

When you post to the MyID authorization URL, set the `client_id` to the ID of your intranet system; for example:

```
myid.intranet
```

You set up this ID when you configured the authentication server; see section [5.1](#), *Configuring web.oauth2 for user-based authentication*.

3. Use the authorization code to request an access token, passing the PKCE code verifier.

See the *Requesting an access token* section.

Once you have carried out this procedure, you will have a block of JSON containing an `access_token` that you can then use to authenticate to your embedded Operator Client screen.

5.2.1 Example requests

On your server, create the following pages.

The first page is `default.asp` – this page requests the authorization code.

```
<html>
<head>
  <title>Request authorization</title>
</head>
<body>
  <form method=post enctype="application/x-www-form-
urlencoded" action="https://react.domain31.local/web.oauth2/connect/authorize">
    <p>Client id: <input type="text" name="client_id" value="myid.intranet"></p>
    <p>Scope: <input type="text" name="scope" value="myid.rest.basic"></p>
    <p>Redirect: <input type="text" name="redirect_
uri" value="https://react.domain31.local/mysystem/callback.asp"></p>
    <input type="hidden" name="response_type" value="code">
    <input type="hidden" name="code_challenge" value="lzKaVv4bWu06z_
m0yFynJj6zttnU5gYpXah8tLYKzGg">
    <input type="hidden" name="code_challenge_method" value="S256">
    <input type="submit">
  </form>
</body>
</html>
```

This page contains a simple form that calls the authorization endpoint.

- The `client_id` is set to `myid.intranet` – this must match the entry you added to the `appsettings.Production.json` file.
- The `redirect_uri` is set to `https://react.domain31.local/mysystem/callback.asp` – this must also be included in the `appsettings.Production.json` file.

This is the page to which the server will return the authorization code.

- The `code_challenge` is set to `lzKaVv4bWu06z_m0yFynJj6zttnU5gYpXah8tLYKzGg`, which is the Base64 URL encoded SHA256 hash of the code verifier. The code challenge and the code verifier make up a pair of values that are used to ensure that the same person makes the call to the authorization endpoint and the token endpoint.

The second page is `callback.asp` – this page is passed the authorization code by the authentication server, and then allows you to request the access token.

```
<html>
<head>
  <title>Request access token</title>
</head>
<body>
  <form method=post enctype="application/x-www-form-
urlencoded" action="https://react.domain31.local/web.oauth2/connect/token">
    <input type="hidden" name="grant_type" value="authorization_code">
    <p>Client id: <input type="text" name="client_id" value="myid.intranet"></p>
    <input type="hidden" name="code_
verifier" value="TiGVEDHIRkdTpif4zLw8v6tcdG2VJXvP4r0fuLhsXIj">
    <p>Code: <input type="text" name ="code" value ="<%
response.write(request.querystring("code"))
%>"</p>
    <p>Redirect: <input type="text" name="redirect_
uri" value="https://react.domain31.local/mysystem/callback.asp"></p>
    <input type="submit">
  </form>
</body>
</html>
```

This page is passed the authorization code, then includes this in a simple form to request the access token.

- The `client_id` is set to `myid.intranet` – this must match the entry you added to the `appsettings.Production.json` file.
- The `code_verifier` is set to `TiGVEDHIRkdTpif4zLw8v6tcdG2VJXvP4r0fuLhsXIj` – this is the companion piece to the PKCE code challenge.
- The `code` is set to the authorization code, which is passed to this page in the `code` part of the query string.
- The `redirect_uri` is set to `https://react.domain31.local/mysystem/callback.asp` – this must also be included in the `appsettings.Production.json` file.

5.3 Posting the access token

Once you have an access token, you can post the token to the embedded Operator Client screen, and you are automatically authenticated to that screen until the token expires. If you pass an expired or invalid token, you can authenticate using the **Sign In** option.

Sample code for posting the token:

```
<html>
  <head>
    <script>
      function applyToken() {
        document.getElementById("OCFrame").src = document
          .getElementById("link")
          .value.replace("#/", "#/embedded/");
        document.getElementById("OCFrame").contentWindow.postMessage({
          token: document.getElementById("token").value,
        });
      }
    </script>
    <style>
      #OCFrame {
        border: 3px solid blue;
        width: 80%;
        height: 90%;
      }
      #link,
      #token {
        margin: 2px;
        width: 900px;
      }
    </style>
  </head>
  <body>
    This is Operator Client embedded in an iframe:<br />
    <iframe
      id="OCFrame"
      src=https://react.domain31.local/myid/operatorclient/#
    ></iframe>
    <br />Token: <input type="text" id="token" />
    <br />URL:
    <input
      type="text"
      id="link"
      value=""
    />
    <br />
    <input type="button" value="Apply" onclick="applyToken()" />
  </body>
</html>
```

When you click **Apply**, this simple example sets the location of the iframe to the URL you provided in the form. The code automatically adds `/embedded` to the URL after the `#` to hide the category and search panels.

It then takes the token you provided, and uses `postMessage` to pass the token to the embedded Operator Client screen, automatically authenticating you, assuming the access token is valid.

Important: The `postMessage` method allows cross-origin communication between window objects (for example, between a window and its embedded iframe) and you are encouraged to review the security guidelines for this scenario; for example:

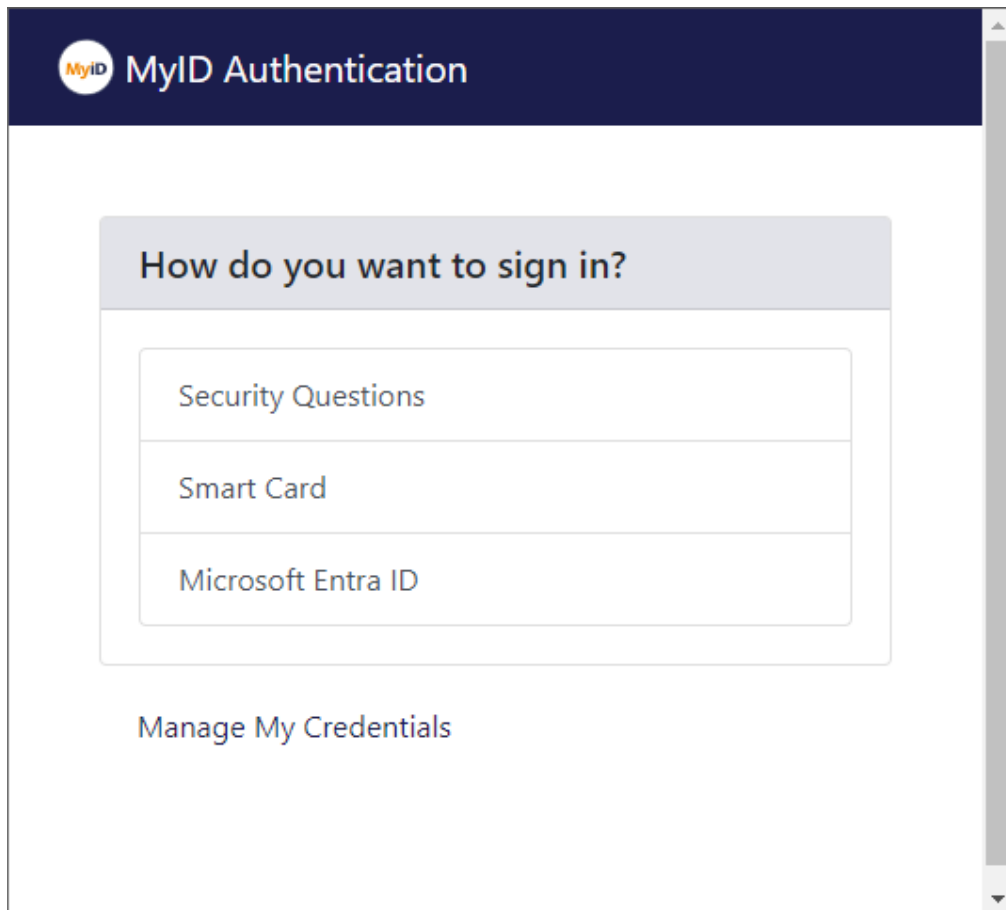
developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

You must make sure that this page is on the same server as MyID.

6 Setting up an external identity provider

You can configure MyID to set up an external OpenID Connect identity provider (for example, Microsoft Entra or Google) to provide authentication to the MyID Operator Client or any other system that uses the MyID web.oidc authentication service.

You can then select the external identity provider from the MyID Authentication screen.



Note: You cannot use external identity providers for MyID Desktop or the Self-Service App; you can use them only for the MyID Operator Client or other systems that you have configured to use the MyID web.oidc authentication service.

You can configure MyID to add new users from your external identity provider, to accept users only if they already exist in MyID, or to update existing users with details from the external identity provider. You can map the information available as claims from the external identity providers to MyID user attributes.

You can:

- Configure Microsoft Entra as an external identity provider.
See section [6.1, *Configuring Microsoft Entra*](#).
- Configure any OpenID Connect system as an external identity provider.
See section [6.2, *Configuring OpenID Connect*](#).
- Configure other types of external identity provider.
See section [6.3, *Configuring other types of identity provider*](#).
- Map attributes from the external identity provider to MyID attributes.
See section [6.4, *Mapping attributes*](#).

6.1 Configuring Microsoft Entra

You can use Microsoft Entra as an external identity provider for MyID.

6.1.1 Configuring Microsoft Entra as an external identity provider

When you configure Microsoft Entra, you can register a new application to use as the external identity provider in MyID.

You are recommended to create the application as a single tenant system, where only members of your own organization have access to MyID; if you create a multitenant system, anyone with an organizational Microsoft account can access your system.

You must configure the following in Microsoft Entra:

- **Redirect URIs**

Add the redirect URI for your web.oauth2 server. This is in the format:

```
https://<server>/web.oauth2/signin-microsoft
```

Where <server> is the address of your MyID web server.

For example:

```
https://myid.mydomain.com/web.oauth2/signin-microsoft
```

- **Client secrets**

Create a client secret in Microsoft Entra. You must take a note of this client secret when you create it, as for security reasons it is not available once you navigate away from the screen on which it is first displayed.

Description	Expires	Value ⓘ	Copy to clipboard	Get ID
MyID Logon	8/20/2024	5hb8Q~v63phpxqZ4EQKv...		7f0f6e27-2c72-43e7-be53...

Note: By default, client secrets created in Microsoft Entra expire after 180 days. You must make sure you set up procedures to remind you to create a new client secret before the current client secret expires, and to update your MyID web.oauth2 configuration with the new client secret.

- **Application (client) ID**

Take a note of the client ID. You need this when configuring the web.oauth2 server.

- **Directory (tenant) ID**

Take a note of the tenant ID. You need this to specify the Entra authorization and token endpoints when configuring the web.oauth2 server.

6.1.2 Encrypting the client secret

Because you are going to store the client secret in a configuration text file on the server, you must encrypt it for security purposes. MyID supports DPAPI for encrypting the client secret; this uses the logged-on Windows user (in this case, the MyID web service user) to encrypt the secret, and only the same Windows user can decrypt the secret.

To encrypt the client secret:

1. On the MyID web server, log on as the user under which the web.oauth2 service runs.

By default, this is the MyID Web Service user; you can confirm this by checking which user is configured for the **myid.web.oauth2.pool** application pool in IIS.

Note: It is important that you use this account to encrypt the secret, as no other accounts can decrypt the secret to use it.

2. Open a Windows PowerShell command prompt, and navigate to the web.oauth2 folder.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2\
```

3. Run the following PowerShell script:

```
.\DPAPIEncrypt.ps1 <secret>
```

For example:

```
.\DPAPIEncrypt.ps1 b5989015-bb9e-4533-874b-2b4a6a8280ed
```

The script outputs an encrypted copy of the secret; for example:

```
PS C:\Program Files\Intercede\MyID\web.oauth2> .\DPAPIEncrypt.ps1  
b5989015-bb9e-4533-874b-2b4a6a8280ed  
AQAAANCMnd8BFdERjHoAwE/C [...] JwWwaKXWoS3i+ulxtmjVQyudpQ==
```

(Encrypted output string truncated for documentation purposes.)

4. Copy the encrypted secret.

6.1.3 Configuring the web.oauth2 server for Microsoft Entra

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service.

2. Add an entry to the `ExternalProviders` array.

If the `ExternalProviders` array does not exist, add it at the top level of the file. For reference, the `appsettings.json` file contains an empty `ExternalProviders` array that allows you to confirm its location.

```
"ExternalProviders":[
  {
    "Name":"<name>",
    "LogonMechanismId":101,
    "Action":"<action>",
    "MicrosoftAccountOptions":{
      "ClientId":"<client ID>",
      "ClientSecret":"<client secret>",
      "AuthorizationEndpoint":"https://login.microsoftonline.com/<tenant
ID>/oauth2/v2.0/authorize",
      "TokenEndpoint":"https://login.microsoftonline.com/<tenant
ID>/oauth2/v2.0/token"
    },
    "ClientSecretEncrypted":"<encrypted client secret>",
    "Mappings":[
      {}
    ]
  }
]
```

where:

- `<name>` – the label used for the authentication method in the MyID Authentication dialog.
- `<action>` – one of the following actions:
 - `Find` – (default) the user must already exist in the MyID database; the claims from the external identity provider must identify a user already in the MyID database.
 - `Create` – if the user does not already exist in MyID, they are created. If the user does exist, that same user is used.
 - `Update` – the user must already exist in MyID. Mapped fields marked with `Update:true` are updated in MyID based on the value supplied by the external identity provider.

See section 6.4, *Mapping attributes* for details of setting the update options for mapped fields.

- `CreateAndUpdate` – if the user does not already exist in MyID, they are created. If the user does exist, that same user is used and any fields marked with `Update:true` are updated.
 - `<client ID>` – the client ID from your Microsoft Entra configuration.
 - `<client secret>` – the client secret you created in Microsoft Entra.
- Important:** For production systems, you are recommended *not* to include the client secret in the `appsettings.production.json` file, but to encrypt the client secret and use the `ClientSecretEncrypted` option instead.
- `<tenant ID>` – the tenant ID from your Microsoft Entra configuration. You must include the tenant ID in both the `AuthorizationEndpoint` and `TokenEndpoint` options.
- Note:** If you are using a multitenant system, you do not need to include the `AuthorizationEndpoint` and `TokenEndpoint` options.
- `<encrypted client secret>` – the encrypted client secret. See section 6.1.2, [Encrypting the client secret](#).

Note: The `LogonMechanismId` is set to 101, which is fixed for Microsoft Entra.

For example:

```
"ExternalProviders":[
  {
    "Name":"Microsoft Entra ID",
    "LogonMechanismId":101,
    "Action":"CreateAndUpdate",
    "MicrosoftAccountOptions":{
      "ClientId":"bb61c9f6-9a71-42ba-a156-05db9a7a6407",
      "ClientSecret":""
    },
    "AuthorizationEndpoint":"https://login.microsoftonline.com/2fad39ef-cead-489d-a755-c3b45c762c4a/oauth2/v2.0/authorize",
    "TokenEndpoint":"https://login.microsoftonline.com/2fad39ef-cead-489d-a755-c3b45c762c4a/oauth2/v2.0/token"
  },
  "ClientSecretEncrypted":"AQAAANCMnd8BFdERjHoAwE/C [...]
  JwWwaKXWoS3i+u1xtmjVQyudpQ==",
  "Mappings":[
    {}
  ]
}
```

For further information about the settings available in the `MicrosoftAccountOptions` section, see the Microsoft documentation for the ASP.NET Core `MicrosoftAccountOptions` class:

learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.microsoftaccount.microsoftaccountoptions?view=aspnetcore-8.0

6.1.4 Configuring MyID to use Microsoft Entra

Once you have set up Microsoft Entra, and configured the web.oidc server, you can configure MyID to make Microsoft Entra available as a logon mechanism.

To enable the Microsoft Entra logon mechanism:

1. In the MyID Operator Client, from the **More** category, select **Configuration Settings > Security Settings**.

Alternatively, in MyID Desktop, from the **Configuration** category, select the **Security Settings** workflow.

2. Click the **Logon Mechanisms** tab.
3. Set the following option:
 - **Microsoft Entra ID** – set to **Yes** to allow logon using Microsoft Entra.
4. Click **Save changes**.

To make the Microsoft Entra logon mechanism available to users:

1. In the MyID Operator Client, from the **More** category, select **Configuration Settings > Edit Roles**.

Alternatively, in MyID Desktop, from the **Configuration** category, select the **Edit Roles** workflow.

2. Click **Logon Methods**.
3. In the Logon Mechanisms dialog, select the **Microsoft Entra ID** logon mechanism for each role you want to be able to log on using Microsoft Entra.
4. Click **OK**, then click **Save Changes**.

6.1.5 Next steps

You can now map the attributes from the external identity provider to MyID attributes. See section [6.4, Mapping attributes](#).

See also section [6.5, Example Microsoft Entra settings](#) for a sample set of mappings for Microsoft Entra.

6.2 Configuring OpenID Connect

You can use any OpenID Connect server as an external identity provider for MyID.

Note: If you are using Google, you can use a streamlined configuration process; see section [6.3, Configuring other types of identity provider](#).

6.2.1 Configuring OpenID Connect as an external identity provider

You must configure your OpenID Connect system to allow the MyID web.oauth2 service to connect to it. The specific instructions for each system are different; see your server's documentation for details.

This section provides general principles for configuring an OpenID Connect system.

You must configure the following in your OpenID Connect system:

- **Redirect URIs**

Add the redirect URI for your web.oauth2 server. This is in the format:

```
https://<server>/web.oauth2/<identifier>
```

Where:

- <server> is the address of your MyID web server.
- <identifier> is an identifier for the external identity provider.

Make sure the <identifier> corresponds to the `CallbackPath` value you set up for the web.oauth2 server (see section [6.2.3, Configuring the web.oauth2 server for OpenID Connect](#)).

For example:

```
https://myid.mydomain.com/web.oauth2/loginOidc121
```

- **Client secrets**

Create a client secret in your OpenID Connect system. You must take a note of this client secret when you create it.

- **Client ID**

Take a note of the client ID. You need this when configuring the web.oauth2 server.

- **Configuration endpoint**

Find the URL that hosts the OpenID Connect configuration well-known endpoint.

This is often of the form `https://<host>/<id>/v2.0/.well-known/openid-configuration`.

You need this when configuring the web.oauth2 server.

6.2.2 Encrypting the client secret

Because you are going to store the client secret in a configuration text file on the server, you must encrypt it for security purposes. MyID supports DPAPI for encrypting the client secret; this uses the logged-on Windows user (in this case, the MyID web service user) to encrypt the secret, and only the same Windows user can decrypt the secret.

To encrypt the client secret:

1. On the MyID web server, log on as the user under which the web.oauth2 service runs.

By default, this is the MyID Web Service user; you can confirm this by checking which user is configured for the **myid.web.oauth2.pool** application pool in IIS.

Note: It is important that you use this account to encrypt the secret, as no other accounts can decrypt the secret to use it.

2. Open a Windows PowerShell command prompt, and navigate to the web.oauth2 folder.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2\
```

3. Run the following PowerShell script:

```
.\DPAPIEncrypt.ps1 <secret>
```

For example:

```
.\DPAPIEncrypt.ps1 b5989015-bb9e-4533-874b-2b4a6a8280ed
```

The script outputs an encrypted copy of the secret; for example:

```
PS C:\Program Files\Intercede\MyID\web.oauth2> .\DPAPIEncrypt.ps1  
b5989015-bb9e-4533-874b-2b4a6a8280ed  
AQAAANCMnd8BFdERjHoAwE/C [...] JwWwaKXWoS3i+ulxtmjVQyudpQ==
```

(Encrypted output string truncated for documentation purposes.)

4. Copy the encrypted secret.

6.2.3 Configuring the web.oauth2 server for OpenID Connect

1. In a text editor, open the `appsettings.Production.json` file for the web service.

By default, this is:

```
C:\Program Files\Intercede\MyID\web.oauth2\appsettings.Production.json
```

This file is the override configuration file for the `appsettings.json` file for the web service.

2. Add an entry to the `ExternalProviders` array.

If the `ExternalProviders` array does not exist, add it at the top level of the file. For reference, the `appsettings.json` file contains an empty `ExternalProviders` array that allows you to confirm its location.

```
"ExternalProviders":[
  {
    "Name": "<name>",
    "LogonMechanismId": <logon mechanism>,
    "Action": "<action>",
    "OpenIdConnectOptions": {
      "ClientId": "<client ID>",
      "ClientSecret": "<client secret>",
      "Authority": "<authority URL>",
      "ResponseType": "code",
      "GetClaimsFromUserInfoEndpoint":true,
      "Scope": [<scopes>],
      "CallbackPath": "<callback path>",
      "Prompt": "login"
    },
    "ClientSecretEncrypted":"<encrypted client secret>",
    "Mappings":[
      {}
    ]
  }
]
```

where:

- `<name>` – the label used for the authentication method in the MyID Authentication dialog.
- `<logon mechanism>` – the ID of the logon mechanism. MyID provides the following logon mechanism IDs that you can use for OpenID Connect systems:
 - 121 – corresponds to **External IDP 1**.
 - 122 – corresponds to **External IDP 2**.
 - 123 – corresponds to **External IDP 3**.
- `<action>` – one of the following actions:
 - `Find` – (default) the user must already exist in the MyID database; the claims from the external identity provider must identify a user already in the MyID database.
 - `Create` – if the user does not already exist in MyID, they are created. If the user does exist, that same user is used.

- `Update` – the user must already exist in MyID. Mapped fields marked with `Update:true` are updated in MyID based on the value supplied by the external identity provider.
See section 6.4, *Mapping attributes* for details of setting the update options for mapped fields.
- `CreateAndUpdate` – if the user does not already exist in MyID, they are created. If the user does exist, that same user is used and any fields marked with `Update:true` are updated.
- `<client ID>` – the client ID from your Microsoft Entra configuration.
- `<client secret>` – the client secret you created in Microsoft Entra.
Important: For production systems, you are recommended *not* to include the client secret in the `appsettings.production.json` file, but to encrypt the client secret and use the `ClientSecretEncrypted` option instead.
- `<authority URL>` – the URL that hosts the OpenID configuration well-known endpoint. For example, for Microsoft Entra, this is:

```
https://login.microsoftonline.com/<tenant id>/v2.0
```
- `<scopes>` – an array of scopes that you want to obtain from the external identity provider.
For example:

```
"Scope": ["openid", "email", "profile", "user.read"],
```

Note: `user.read` is an Entra-specific scope that allows more user data to be returned.
- `<callback path>` – the path to which the response is returned. This must correspond to the redirect URI you set up.
For example, set the callback path to:

```
/loginOidc121
```


If your server is `myid.mydomain.com`, the response is returned to:

```
https://myid.mydomain.com/web.oauth2/loginOidc121
```
- `<encrypted client secret>` – the encrypted client secret. See section 6.2.2, *Encrypting the client secret*.

For example:

```
"ExternalProviders":[
  {
    "Name":"OpenID Connect",
    "LogonMechanismId":121,
    "Action":"CreateAndUpdate",
    "OpenIdConnectOptions":{
      "ClientId":"bb61c9f6-9a71-42ba-a156-05db9a7a6407",
      "ClientSecret":"",
      "Authority":"https://login.microsoftonline.com/2fad39ef-cead-489d-
a755-c3b45c762c4a/v2.0",
      "ResponseType":"code",
      "GetClaimsFromUserInfoEndpoint":true,
      "Scope":[
        "openid",
        "email",
        "profile",
        "user.read"
      ],
      "CallbackPath":"/loginOidc121",
      "Prompt":"login"
    },
    "ClientSecretEncrypted":"AQAAANCMnd8BFdERjHoAwE/C [...]
JwWwaKXWoS3i+ulxtmjvQyudpQ==",
    "Mappings":[
      {}
    ]
  }
]
```

For further information about the settings available in the `OpenIdConnectOptions` section, see the Microsoft documentation for the ASP.NET Core `OpenIdConnectOptions` class:

learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.openidconnectoptions?view=aspnetcore-1.1&viewFallbackFrom=aspnetcore-8.0

6.2.4 Configuring MyID to use OpenID Connect

Once you have set up your OpenID Connect server, and configured the web.oidc server, you can configure MyID to make OpenID Connect available as a logon mechanism.

To enable the OpenID Connect logon mechanism:

1. In the MyID Operator Client, from the **More** category, select **Configuration Settings > Security Settings**.

Alternatively, in MyID Desktop, from the **Configuration** category, select the **Security Settings** workflow.

2. Click the **Logon Mechanisms** tab.
3. Set one of the following options:
 - **External IDP 1** – corresponds to logon mechanism ID 121.
 - **External IDP 2** – corresponds to logon mechanism ID 122.
 - **External IDP 3** – corresponds to logon mechanism ID 123.

Use the logon mechanism that corresponds to the ID you specified in the `LogonMechanismId` field in the configuration file.

4. Click **Save changes**.

To make the OpenID Connect logon mechanism available to users:

1. In the MyID Operator Client, from the **More** category, select **Configuration Settings > Edit Roles**.

Alternatively, in MyID Desktop, from the **Configuration** category, select the **Edit Roles** workflow.

2. Click **Logon Methods**.
3. In the Logon Mechanisms dialog, select the appropriate **External IDP** logon mechanism for each role you want to be able to log on using your OpenID Connect system.
4. Click **OK**, then click **Save Changes**.

6.2.5 Next steps

You can now map the attributes from the external identity provider to MyID attributes. See section [6.4, Mapping attributes](#).

See also section [6.6, Example OpenID Connect settings](#) for a sample set of mappings for OpenID Connect.

6.3 Configuring other types of identity provider

MyID supports any external identity provider that conforms to OpenID Connect (see section [6.2, Configuring OpenID Connect](#)), but also provides streamlined configuration for specific types of identity provider where there is a class to support their configuration.

For example:

- Google

Instead of using the `OpenIdConnectOptions` section in your configuration file, you can use `GoogleOptions`, which is preconfigured for a Google external identity provider.

In most situations, you need only set the following options:

- `ClientId`
- `ClientSecretEncrypted` (or `ClientSecret`)

When setting the redirect URI in the external identity provider, use:

```
https://<server>/web.oauth2/signin-google
```

Where:

- `<server>` is the address of your MyID web server.

For example:

```
https://myid.mydomain.com/web.oauth2/signin-google
```

For a full list of configuration options, see:

learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.builder.googleoptions?view=aspnetcore-1.1&viewFallbackFrom=aspnetcore-8.0

Mapping attributes works in the same way for all identity providers; see section [6.4, Mapping attributes](#).

6.4 Mapping attributes

Once you have configured your external identity provider, you must configure MyID to process the attributes returned in the claims, and map them to MyID attributes.

6.4.1 Finding a list of available claims

To determine which claims are returned from your external identity provider, you can enable logging for the web.oidc service.

See the *MyID REST and authentication web services* section in the [Configuring Logging](#) guide for details of enabling logging. Set the log level to `INFO`.

Once you have edited the `Log.config` file, add the following minimal `Mappings` section to the entry for your external identity provider in the `ExternalProviders` array of the `appsettings.Production.json` file for the web.oidc service:

```
"Mappings":[
  {
    "Match":{
    },
    "Attributes":[
      {
        "From":"givenName",
        "To":"FirstName"
      },
      {
        "From":"surname",
        "To":"Surname"
      },
      {
        "From":"",
        "To":"Roles",
        "Static":"<role name='Cardholder' scope='1' />"
      },
      {
        "From":"",
        "To":"GroupName",
        "Static":"Imported From External IDP"
      },
      {
        "From":"",
        "To":"ParentGroupName",
        "Static":"External"
      }
    ]
  }
]
```

This set of mappings provides a minimal set of data to allow the web.oidc service to attempt to carry out a logon using the external identity provider; it does not allow you to log on, but allows MyID to retrieve the claims information from the external identity provider.

Save the file, then recycle the application pool in IIS to ensure that the web service is using the latest settings.

1. On the MyID web server, in Internet Information Services (IIS) Manager, select **Application Pools**.
2. Right-click the **myid.web.oauth2.pool** application pool, then from the pop-up menu click **Recycle**.

You can now attempt to log on to MyID using the external identity provider. The attempt fails with error OA10077, but the log displays a list of the claims returned from the identity provider that you can use to set up your mappings.

For example:

```
<LogEntry Type="InfoMessage">
  <TimeStamp>2024-02-22 14:42:15.560</TimeStamp>
  <ManagedThreadId>5</ManagedThreadId>
  <Class>Intercede.MyID.Logging.Log4Net.Log4NetLogger</Class>
  <Method>Microsoft.Extensions.Logging.ILogger.Log</Method>
  <Message>List of all claims available from provider: @odata.context:
https://graph.microsoft.com/v1.0/$metadata#users/$entity,
  businessPhones: ["259"],
  displayName: Susan Smith,
  givenName: Susan,
  jobTitle: Business Analyst,
  mail: Susan.Smith@example.com,
  mobilePhone: ,
  officeLocation: Headquarters,
  preferredLanguage: ,
  surname: Smith,
  userPrincipalName: Susan.Smith@example.com,
  id: b0e777c5-02ff-4669-9c99-18780b334bd7</Message>
</LogEntry>
```

6.4.2 Matching attributes

You can use the `Match` option to restrict the users who can authenticate using the external identity provider by specifying a value for one of the claims returned.

For example, you may want to allow access only to people where the `officeLocation` claim is set to `Headquarters`:

```
"Match": {
  "officeLocation": "Headquarters"
},
```

6.4.3 Mapping attributes

You can include multiple items in the `Attributes` array. Each item can have the following properties:

- `From`

The name of the claim in the external identity provider. The available claims are different for each identity provider; see section [6.4.1, Finding a list of available claims](#) for assistance in determining which claims are available to you.

- `To`

The name of the field in MyID to which to map the attribute. This must match a column name in the `vPeopleUserAccounts` view in the MyID database.

If you are creating a user in MyID, you must provide mappings for the `FirstName`, `Surname`, or both fields in MyID. You must also provide the `FullName`; for example:

```
{
  "From": "displayName",
  "To": "FullName"
},
{
  "From": "givenName",
  "To": "FirstName"
},
{
  "From": "surname",
  "To": "Surname"
},
}
```

- `Mandatory`

If set to `true`, this claim must be present in the external information for the user to be valid.

- Unique

If set to `true`, this value must be unique in MyID for the user to be valid.

Important: This is used as the primary lookup value when searching for the user in MyID; you *must* include at least one `Unique` attribute.

You are recommended to use a unique reference (such as a GUID) supplied from the external identity provider; you can store this value in a MyID attribute to create a link between the user in the external identity provider and MyID.

MyID provides the following fields:

- `XuSYSExternalReferenceId1`
- `XuSYSExternalReferenceId2`
- `XuSYSExternalReferenceId3`

that you can use to store the unique ID. If you have up to three external identity providers, you can use a different field for each.

For example:

```
{
  "From": "id",
  "To": "XuSYSExternalReferenceId1",
  "Mandatory": true,
  "Unique": true,
  "Update": true
},
```

- Update

If set to `true`, and the `Action` configured for the provider is `Update` or `CreateAndUpdate`, MyID uses the information from the claim to update the user record in the MyID database.

- `LookupExisting`

If the primary lookup value (that is, the claim set to `Unique`) is unable to find a match, this acts as a fallback option. If there are still no matching users for `LookupExisting`, the operation moves on to create a new user or to fail logon, depending on the `Action` setting. This feature is intended for the case where a user may have been created not from the external identity provider (so the `Unique` reference is not yet set), but you want to look up an existing user in MyID (for example, by `UserPrincipalName` or `Email`) and then update the `Unique` reference to create the permanent link.

For example:

```
{
  "From": "userPrincipalName",
  "To": "UserPrincipalName",
  "LookupExisting": true,
  "LdapSync": true
}
```

- LdapSync

If set to `true`, the search operation also checks the LDAP for valid matching users to import additional user data from the directory, using the field specified in the `To` property and the value specified for this attribute. The value of the `To` property must be a value in the `LDAPLookUp` table.

Important: If you enable `LookupExisting` or `LdapSync` on an attribute, you must be certain that the source of that data from the external identity provider is trustworthy. If you use these features and the source of the mapped attribute used for `LookupExisting` or `LdapSync` can be controlled by the end user or another untrusted individual, it can enable the user authenticating with that identity provider to impersonate a user in MyID, either by assigning the external identity provider authentication mechanism to that existing user account or by importing data (such as the DN) from the LDAP that belongs to another person who is looked up by that attribute.

- Static

If there is extra information that you needs to provide that does not come from the external provider, such as MyID roles and groups, you can include the information in a `Static` attribute.

If you are creating a user in MyID, you must provide static mappings for the roles you want to set for the user, and a group into which you want to place the user.

To set roles for a user, you must provide XML that describes the roles and scope you want to provide. You can provide multiple roles. Use the following format:

```
<role name ='name' scope='scope' />
```

where:

- `name` – the name of the role from the `Name` field in the `UserProfiles` table in the MyID database.
- `scope` – the ID of the scope. You can use the following:
 - 1 – Self
 - 2 – Department
 - 3 – Division
 - 4 – All

For example:

```
{
  "From": "",
  "To": "Roles",
  "Static": "<role name='Cardholder' scope='1' /><role name='PasswordUser
scope='2' />"
}
```

To set a group for a user, you must set the `To` attribute to `GroupName` and the `Static` value to the name of the group. To set the parent group of the group you are setting for a user, you must set the `To` attribute to `ParentGroupName`, and the `Static` value to the name of the parent group.

For example:

```
{
  "From": "",
  "To": "GroupName",
  "Static": "Imported Group"
},
{
  "From": "",
  "To": "ParentGroupName",
  "Static": "External"
}
```

6.5 Example Microsoft Entra settings

The following is an example set of configuration options and mapping attributes for Microsoft Entra. You can use this as a starting point for your own configuration in the `appsettings.Production.json` file for the `web.oauth2` service; see section 6.1, [Configuring Microsoft Entra](#).

```
"ExternalProviders":[
  {
    "Name":"Microsoft Entra ID",
    "LogonMechanismId":101,
    "Action":"CreateAndUpdate",
    "MicrosoftAccountOptions":{
      "ClientId":"bb61c9f6-9a71-42ba-a156-05db9a7a6407",
      "ClientSecret":"",
      "AuthorizationEndpoint":"https://login.microsoftonline.com/2fad39ef-cead-489d-a755-c3b45c762c4a/oauth2/v2.0/authorize",
      "TokenEndpoint":"https://login.microsoftonline.com/2fad39ef-cead-489d-a755-c3b45c762c4a/oauth2/v2.0/token"
    },
    "ClientSecretEncrypted":"AQAAANCMnd8BFdERjHoAwE/C [...] JwWwaKXWoS3i+ulxtmjVQyudpQ==",
    "Mappings":[
      {
        "Match":{
          "officeLocation":"Headquarters"
        },
        "Attributes":[
          {
            "From":"id",
            "To":"XuSYSExternalReferenceId1",
            "Mandatory":true,
            "Unique":true,
            "Update":true
          },
          {
            "From":"displayName",
            "To":"FullName"
          },
          {
            "From":"givenName",
            "To":"FirstName"
          },
          {
            "From":"surname",
            "To":"Surname"
          },
          {
            "From":"userPrincipalName",
            "To":"UserPrincipalName",
            "LookupExisting":true,
            "LdapSync":true
          },
          {
            "From":"mail",
            "To":"Email"
          },
          {
            "From":"","
            "To":"Roles",

```

```
scope='2'/>"
  "Static": "<role name='Cardholder' scope='1'/><role name='PasswordUser
  },
  {
    "From": "",
    "To": "GroupName",
    "Static": "Imported From Microsoft"
  },
  {
    "From": "",
    "To": "ParentGroupName",
    "Static": "External"
  }
]
}
]
}
```


6.6 Example OpenID Connect settings

The following are example sets of configuration options and mapping attributes for OpenID Connect. You can use these as a starting point for your own configuration in the `appsettings.Production.json` file for the `web.oauth2` service; see section 6.2, [Configuring OpenID Connect](#).

6.6.1 OpenID Connect settings for Microsoft Entra

```
"ExternalProviders":[
  {
    "Name":"OpenID Connect",
    "LogonMechanismId":121,
    "Action":"CreateAndUpdate",
    "OpenIdConnectOptions":{
      "ClientId":"bb61c9f6-9a71-42ba-a156-05db9a7a6407",
      "ClientSecret":"","
      "Authority":"https://login.microsoftonline.com/2fad39ef-cead-489d-a755-
c3b45c762c4a/v2.0",
      "ResponseType":"code",
      "GetClaimsFromUserInfoEndpoint":true,
      "Scope":[
        "openid",
        "email",
        "profile",
        "user.read"
      ],
      "CallbackPath":"/loginOidc121",
      "Prompt":"login"
    },
    "ClientSecretEncrypted":"AQAAANCMnd8BFdERjHoAwE/C [...] JwWwaKXWoS3i+u1xtmjVQyudpQ==",
    "Mappings":[
      {
        "Match":{

        },
        "Attributes":[
          {
            "From":"oid",
            "To":"XuSYSExternalReferenceId2",
            "Mandatory":true,
            "Unique":true,
            "Update":true
          },
          {
            "From":"name",
            "To":"FullName"
          },
          {
            "From":"given_name",
            "To":"FirstName"
          },
          {
            "From":"family_name",
            "To":"Surname"
          },
          {
            "From":"","
```

```
        "To": "Roles",
        "Static": "<role name='Cardholder' scope='1' />"
    },
    {
        "From": "",
        "To": "GroupName",
        "Static": "Imported From OpenID"
    },
    {
        "From": "",
        "To": "ParentGroupName",
        "Static": "External"
    }
  ]
}
]
```

6.6.2 OpenID Connect settings for Okta

```
"ExternalProviders":[
  {
    "Name":"Okta Dev",
    "LogonMechanismId":122,
    "Action":"CreateAndUpdate",
    "OpenIdConnectOptions":{
      "ClientId":"0obr57pqsgospYEYr8A1",
      "Authority":"https://myownoktadomain.okta.com",
      "ResponseType":"code",
      "GetClaimsFromUserInfoEndpoint":true,
      "Scope":[
        "openid",
        "email",
        "profile"
      ],
      "CallbackPath":"/login0idc122",
      "Prompt":"login"
    },
    "ClientSecretEncrypted":"AQAAANCMnd8BFdERjHoAwE/C [...] JwWwaKXWoS3i+u1xtmjVQyudpQ==",
    "Mappings":[
      {
        "Match":{
          "email_verified":"True"
        },
        "Attributes":[
          {
            "From":"sub",
            "To":"XuSYSExternalReferenceId3",
            "Mandatory":true,
            "Unique":true,
            "Update":true
          },
          {
            "From":"name",
            "To":"FullName",
            "Update":true
          },
          {
            "From":"family_name",
            "To":"Surname",
            "Update":true
          },
          {
            "From":"given_name",
            "To":"FirstName"
          },
          {
            "From":"email",
            "To":"Email",
            "LookupExisting":true,
            "Update":true
          },
          {
            "From":"","
            "To":"Roles",
            "Static":"<role name='Cardholder' scope='1'/>"
          }
        ]
      }
    ]
  }
]
```

```
    },  
    {  
      "From": "",  
      "To": "GroupName",  
      "Static": "Imported From Okta"  
    },  
    {  
      "From": "",  
      "To": "ParentGroupName",  
      "Static": "External"  
    }  
  ]  
}  
]  
]
```

7 Reporting on the authentication database

The `AuthenticationAudits` table in the MyID authentication database contains audited information on any attempts to use the MyID authentication service.

You can use the information in this table to generate reports on the authentication activities of your system.

The `AuthenticationAudits` table has the following format:

Field	Format	Description
Id	int	Automatically incrementing primary key field.
LogonMechanism	nvarchar(20)	The type of credential used for the logon attempt; for example: <ul style="list-style-type: none"> • Fido • Passphrase • SmartcardMCS
UserAccountId	int	The user account relating to the user attempting to log on. This is the <code>UserAccounts.ID</code> field in the main MyID database.
UserAccountObjectId	uniqueidentifier	The user account relating to the user attempting to log on. This is the <code>UserAccounts.ObjectId</code> field in the main MyID database.
DeviceId	int	The device relating to the device being used to log on. This is the <code>Devices.ID</code> field in the main MyID database.
DeviceTypeName	nvarchar(50)	The device type name relating to the device being used to log on. This is the <code>Devices.DeviceTypeName</code> field in the main MyID database.
DeviceSerialNumber	nvarchar(50)	The device serial number relating to the device being used to log on. This is the <code>Devices.SerialNumber</code> field in the main MyID database.
LogonDate	datetime	The date and time that this attempt was recorded.

Field	Format	Description
Status	nvarchar(20)	The status of this logon attempt; for example: <ul style="list-style-type: none">• success• failure
ReadableMessage	nvarchar(max)	For failed attempts to log on, this contains extra information that you can use to diagnose why a logon failed. Error codes are included in this information; see the <i>MyID Operator Client error codes</i> section in the Error Code Reference guide.
LogonName	nvarchar(256)	The logon name of the user attempting to log on. This is the <code>UserAccounts.LogonName</code> field in the main MyID database.
ClientId	nvarchar(100)	The OAuth client Id in use; for example: <ul style="list-style-type: none">• myid.idms• myid.adfs• myid.operatorclient