MyID PIV

Version 12.8

MyID Core API

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

Document reference: IMP2007-12.8.0-PIV

September 2023

Copyright

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

1 Introduction

The MyID® Core API gives you access to the features used in the MyID Operator Client using a REST-based API, allowing you to integrate into other business systems that provide information to MyID or to trigger credential lifecycle events.

The API is secure by default, requiring authentication of the calling system in order to use the API and restricting access to available features and data using MyID role-based access and scope control.

You can use the API for such actions as:

- Searching and retrieving information about a person, device or request.
- Adding or updating a person's information in MyID.
- Managing the lifecycle of people, devices, and requests.

Comprehensive API documentation is provided, including schema information to simplify integration development.

2 Configuring access

This section provides details of configuring access to the API:

- For details of providing access to the built-in API documentation, see section *2.1*, *Accessing the API documentation*.
- For details of configuring access to the features of the API, see section *2.2*, *Accessing the API features*.

2.1 Accessing the API documentation

The API documentation is provided on the API server at the following address:

`https://<myserver>/rest.core/swagger/index.html`

where `<myserver>` is the name of the server hosting the web service.

If necessary, you can configure the API documentation to prevent access. You can also to configure the API documentation to provide details of error codes, and to allow you to use the documentation as a test harness, subject to the appropriate authentication.

To configure access the API documentation:

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

   By default, this is:

   `C:\Program Files\Intercede\MyID\rest.core\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, and include the following:

   ```
   {
     "MyID":  {
       "SwaggerApiDocumentation":  {
         "GenerateDocumentation":  true,
         "DocumentErrorCodes":  false,
         "DocumentMIReports":  true,
         "AllowTestFromDocumentation":  false,
         "ShowPermissions":  false
       }
     }
   }
   ```

   If the `appsettings.Production.json` file already exists, add the above `SwaggerApiDocumentation` section to the file.

2. Edit the following values:

   - `GenerateDocumentation` – set to `false` to prevent access to the documentation.
   - `DocumentErrorCodes` – set to `true` to include details of the web service error messages associated with HTTP status codes.

     Note: If you are consuming the Swagger output using an external tool, you may experience compatibility issues if you set this value to `true`; set `DocumentErrorCodes` to `false` to prevent this.
   - `DocumentMIReports` – set to true to include details of Management Information reports.
   - `AllowTestFromDocumentation` – set to `true` to allow you to use the documentation as a test harness, subject to the appropriate authentication. When you enable this option, the API documentation provides authentication instructions at the top of the page; see also section *5.2, Calling the API from the documentation*.
   - `ShowPermissions` – set to true to display the permissions required for each API call. See section *2.2, Accessing the API features* for more information.

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 **myid.rest.core.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. Open a web browser, and navigate to the following URL:

   `https://<myserver>/rest.core/swagger/index.html`

   where `<myserver>` is the name of the server hosting the web service.

2.2 Accessing the API features

Access to features of the MyID Core API is controlled using MyID roles.

For example, the MyID Operator Client feature that allows you to view a person's images (View Person's Images) is enabled if the operator has a role with one of the following permissions:

- Add Person
- Approve Person
- Edit Person
- Edit PIV Applicant
- Initial PIV Enrollment
- Request Card
- Request Replacement Card
- Update PIV Applicant
- Unapprove Person
- View Person

If the operator account has access to *any* of these permissions, it can use the corresponding API call:

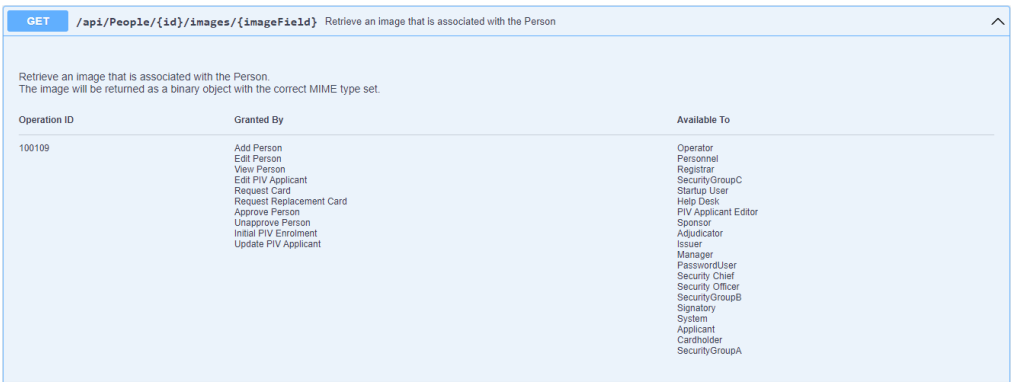
- `GET /api/People/{id}/images/{imageField}`

For information on setting role permissions, see the *Roles* section in the ***Administration Guide***.

Note: As development of the API proceeds in advance of the development of the MyID Operator Client, you may find some API features that do not correspond to Operator Client features. These features do not have role-based restrictions placed on them; however, the object of the operations will always respect the scope of the operator user.

To determine what permissions are required for an API call, set the `ShowPermissions` option to `true` in the rest.core configuration file; see section *2.1*, *Accessing the API documentation* for details.

This adds a section to the API documentation that lists the permissions for each API call, and which roles currently have access:



2.2.1 Scope

The MyID Core API respects the scope of the operator account used to access the API. For example, if you are using an operator account in the Finance department that has a role with a scope of Department, that account can view and access only the people (and their devices, requests, and so on) who are in the Finance department.

For information on setting roles and scope permissions, see the *Scope and security* section in the ***Administration Guide***.

3 Server-to-server authentication

This section provides information on setting up server-to-server authentication for the MyID Core API. This allows you to call the MyID Core API from an external system.

To set up server-to-server authentication, you must carry out the following:

- Configure MyID with a user account that has the appropriate permissions to access the API.

  See section *3.1*, *Configuring MyID for server-to-server authentication*.
- Configure the web.oauth2 web service with the ID of your external system and a shared secret.

  See section *3.2*, *Configuring web.oauth2 for server-to-server authentication*.
- Obtain an access token for your system to use.

  See section *3.3*, *Obtaining a server-to-server access token*.

For more information, see the *Client Credentials* section of the *OAuth 2.0 Authorization Framework*:

tools.ietf.org/html/rfc6749#section-1.3.4

3.1 Configuring MyID for server-to-server authentication

The MyID Core API uses a user account to log on to the MyID system. This allows you to configure access to particular MyID features and groups using the standard roles, groups, and scope feature of MyID. All actions carried out through the API are audited under this user account, and if necessary you can disable the account to prevent access to the API.

3.1.1 Allowing the Client Credentials OAuth2 logon mechanism

To allow access to MyID through server-to-server authentication, you must enable the Client Credentials OAuth2 logon mechanism.

1. In MyID Desktop, from the **Configuration** category, select **Security Settings**.
2. On the **Logon Mechanisms** tab, set the following option:
   - **Client Credentials OAuth2 Logon** – set to Yes to enable server-to-server authentication, or No to disable it.
3. Click **Save changes**.

3.1.2 Creating a role for the external system

You are recommended to create a new role to be used for controlling access to MyID from the external system, rather than using an existing role. This allows you to maintain clear control over the MyID features the external system can access.

To create the role:

1. In MyID Desktop, from the **Configuration** category, select **Edit Roles**.
2. Click **Add**.
3. Give the role a name; for example, `External API`.
4. From the **Derived from** drop-down list, select **Allow None**.
5. Click **Add**.
6. Select the options that relate to the API features you want to be able to access through the API.

   See section *2.2*, *Accessing the API features* for a list of which options map to the API end points.

   You are strongly recommended to select only those options that your external system will need to use.
7. Click **Logon Methods**.
8. For the role you created, select the **Client Credentials OAuth2** logon mechanism, then click **OK**.
9. Click **Save Changes**.

For more information about using the **Edit Roles** workflow, see the *Roles* section in the ***Administration Guide***.

3.1.3 Selecting a group for the user account

Before you create the user account, you must consider into which group you want to put the account. The group you select affects the scope of the user.

- If you want to restrict the access of the API to a particular group of users in MyID, put the API user into the same group, then select a scope of Department or Division when you specify the role for the user account.
- If you do not want to restrict the access of the API, you are recommended to create a separate group for the API user, then select a scope of All when you specify the role. Use the **Add Group** workflow in MyID Desktop to create the group; you can restrict this group to the API role only, and assign this as the default role with a scope of All.

  See the *Adding a group* section in the ***Operator's Guide*** for details of adding groups, and the *Default roles* section in the ***Administration Guide*** for details of setting default roles.

3.1.4 Creating a user account for the external system

Once you have created the role and decided which group to use, you can create the user account that the API will use to access MyID.

To add the user account:

1. In the MyID Operator Client, select the **People** category.
2. Click **ADD**.
3. Provide a **First Name** and **Last Name**; for example, `External API`.
4. Provide a **Logon** name; for example, `api.external`.
5. Select a **Group** for the user account.

   See section *3.1.3*, *Selecting a group for the user account* above for considerations.
6. Select the **Roles** for the user account.

   Select the role you created for use by the external system. Set the appropriate scope; for example, **All** to allow the API to access data related to any user account in the system, or **Division** to restrict access data related to accounts in the same group as the API user, along with any subgroups.
7. Click **SAVE**.

Make sure you take a note of the logon name for the user; you need this for configuring the web.oauth2 web service.

3.2 Configuring web.oauth2 for server-to-server authentication

The MyID authentication web service is called web.oauth2; you must configure this web service to allow access to the API. For server-to-server authentication, you do this using a client credential grant, which you request using a shared secret.

Note: Before you begin, you must decide on a client ID for your external system; for example, `myid.mysystem`. This represents your back-end system that intends to make calls to the MyID Core API.

3.2.1 Creating a shared secret

Important: You must keep the shared secret safe. This information can be used to grant authorization to the API, so must be an unguessable value; for this reason, you are recommended to generate a GUID for the shared secret.

To create a shared secret:

1. Generate a GUID to use as the secret.

   For example:

   ```
   82564d6e-c4a6-4f64-a6d4-cac43781c67c
   ```

2. Create a hash of this GUID using SHA-256, and convert it to Base64.

   For example:

   ```
   kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk=
   ```

   You need this value for the web.oauth2 server. The server does not store the secret, only the hash.

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

3. Combine your client ID, a colon, and the GUID secret:

   For example:

   ```
   myid.mysystem:82564d6e-c4a6-4f64-a6d4-cac43781c67c
   ```

4. Convert this string to Base64.

   For example:

   ```
   bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M=
   ```

   This is the value you will send in the Authorization header for HTTP basic authentication when requesting the access token; alternatively, you can pass the client ID and the shared secret separately in the body as `client_id` and `client_secret` parameters.

3.2.1.1 Example

The following PowerShell example script shows the process for generating a shared secret and creating the hash and Base64 versions you need to configure the web.oauth2 server and request an access token.

```
# Set the client ID of your calling system
$client_id = "myid.mysystem"

# Generate a new GUID to use as the shared secret
$secret = (New-Guid).ToString()

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

# Convert the hashed secret to Base64
$clientSecret = [Convert]::ToBase64String($hashOfSecret)

# Combine the client ID and secret into a single Base64 authorization token
$both = "$client_id`:$secret"
$bytes = [System.Text.Encoding]::UTF8.GetBytes($both)
$combined =[Convert]::ToBase64String($bytes)

# Output the results
Write-Output ("`r`nThe shared secret is: `r`n`r`n$secret")
Write-Output ("`r`nAnd the hash of the shared secret in base64 is:`r`n`r`n$clientSecret" )
Write-Output ("`r`nStore this value in the ClientSecrets of the web.oauth2 appsettings
file.")
Write-Output ("`r`nThe combined string of the client ID and the shared secret
is:`r`n`r`n$both")
Write-Output ("`r`nAnd in base64 this is: `r`n`r`n$combined")
Write-Output ("`r`nUse this value to request an access token from the web service.")

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

Example output:

```
PS C:\Intercede> .\secret.ps1

The shared secret is:

82564d6e-c4a6-4f64-a6d4-cac43781c67c

And the hash of the shared secret in base64 is:

kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk=

Store this value in the ClientSecrets of the web.oauth2 appsettings file.

The combined string of the client ID and the shared secret is:

myid.mysystem:82564d6e-c4a6-4f64-a6d4-cac43781c67c

And in base64 this is:

bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M=

Use this value to request an access token from the web service.

Press any key to continue...
```

3.2.2 Configuring the authentication service

Once you have created a Base64 version of the hash of the shared secret, you can configure the web.oauth2 server.

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. If this file does not already exist, you must create it in the same folder as the `appsettings.json` file.

2. Edit the file to include the following:

```
{
    "Clients":  [
                    {
                       "ClientId":  "<my client ID>",
                       "ClientName":  "<my client name>",
                       "AccessTokenLifetime":  <time>,
                       "AllowedGrantTypes":  [
                                                "client_credentials"
                                             ],
                       "ClientSecrets":  [
                                             {
                                                "Value":  "<secret>"
                                             }
                                          ],
                       "AllowedScopes":  [
```

```
                                        "myid.rest.basic"
                                    ],
                        "Properties":  {
                                         "MyIDLogonName":  "<my user
account>"
                                    }
                    }
        ]
 }
```

where:

- `<my client ID>` – the client ID you decided on; for example:

  `myid.mysystem`

- `<my client name>` – an easily readable name for your client system; for example:

  `My External System`

- `<time>` – the time (in seconds) that the client credential is valid. The default is `3600` – 1 hour.

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

  `kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk=`

- `<my user account>` – the logon name of the user you created to run the API; for example:

  `api.external`

For example:

```
{
    "Clients":  [
                    {
                        "ClientId":  "myid.mysystem",
                        "ClientName":  "My External System",
                        "AccessTokenLifetime":  3600,
                        "AllowedGrantTypes":  [
                                                  "client_credentials"
                                              ],
                        "ClientSecrets":  [
                                              {
"Value":  "kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk="
                                              }
                                          ],
                        "AllowedScopes":  [
                                              "myid.rest.basic"
                                          ],
                        "Properties":  {
"MyIDLogonName":  "api.external"
                                       }
                    }
        ]
 }
```

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

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.

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**.

   This ensures that the web service has picked up the changes to the configuration file.
5. Check that the web.oauth2 server 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 *7*, *Troubleshooting* for information on resolving problems that cause HTTP Error 500.30.

3.3 Obtaining a server-to-server access token

Once you have configured MyID to allow server-to-server access, set up the user account for the API, configured the shared secret, and set up the web.oauth2 web service to recognize your external system, you can request an access token that you can then use to call the API.

3.3.1 Requesting an access token

Request the access token from the following location:

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

POST a request in `application/x-www-form-urlencoded` format.

You must provide the following parameters:

- `grant_type=client_credentials`
- `scope=myid.rest.basic`

You must also provide an `Authorization` header containing "`Basic `" followed by your client ID and shared secret, combined in a single Base64 string.

For example, if your client ID is:

```
myid.mysystem
```

and the secret is:

```
82564d6e-c4a6-4f64-a6d4-cac43781c67c
```

the combination is:

```
myid.mysystem:82564d6e-c4a6-4f64-a6d4-cac43781c67c
```

and the Base64 string is:

```
bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M=
```

and the authorization token is:

```
Basic bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M=
```

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

For example (using cURL):

```
curl -k -i -H "Content-Type: application/x-www-form-urlencoded" -X POST
"https://myserver.example.com/web.oauth2/connect/token" -d "grant_type=client_
credentials&scope=myid.rest.basic" -H "Authorization: Basic
bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M="
```

or using PowerShell:

```
$combined = "bXlpZC5teXN5c3RlbTo4MjU2NGQ2ZS1jNGE2LTRmNjQtYTZkNC1jYWM0Mzc4MWM2N2M="

# Set up the body of the request
$body = @{grant_type='client_credentials'
    scope='myid.rest.basic'
    }
# Set up the header of the request
$header = @{'Content-Type'='application/x-www-form-urlencoded'
    Authorization="Basic $combined"
    }

# Request the access token
Invoke-WebRequest -Method POST -Uri
'https://myserver.example.com/web.oauth2/connect/token' -body $body -Headers $header |
Select-Object -Expand Content

#Wait for a keypress
Write-Host "`r`nPress any key to continue..." -ForegroundColor Yellow

[void][System.Console]::ReadKey($true)
```

An alternative method, passing the `client_id` and `client_secret` in the body rather than in the header:

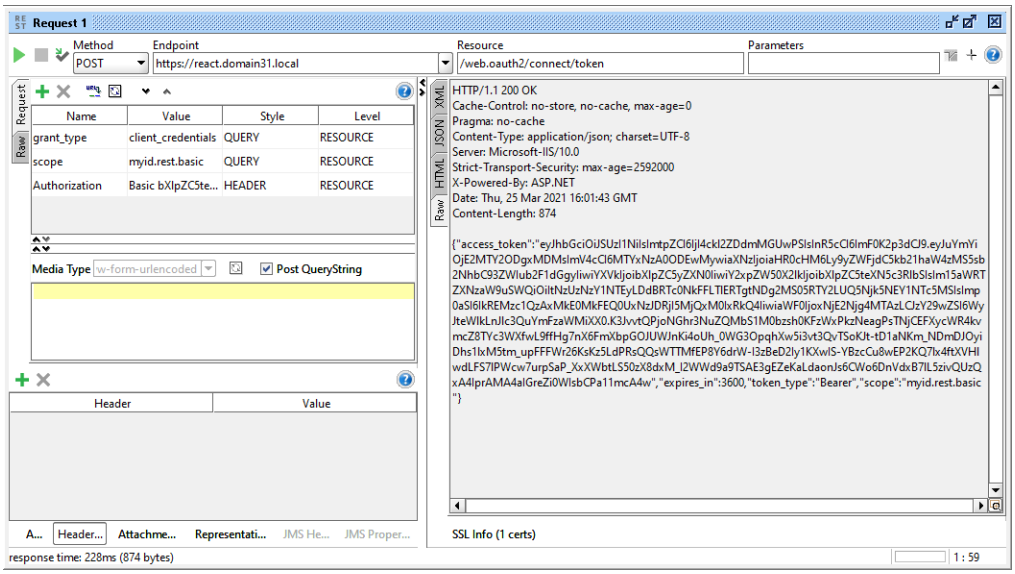
```
# Set up the body of the request
$body = @{grant_type='client_credentials'
    scope='myid.rest.basic'
    client_id='myid.mysystem'
    client_secret='82564d6e-c4a6-4f64-a6d4-cac43781c67c'
    }
# Set up the header of the request
$header = @{'Content-Type'='application/x-www-form-urlencoded'
    }

# Request the access token
Invoke-WebRequest -Method POST -Uri
'https://myserver.example.com/web.oauth2/connect/token' -body $body -Headers $header |
Select-Object -Expand Content

#Wait for a keypress
Write-Host "`r`nPress any key to continue..." -ForegroundColor Yellow

[void][System.Console]::ReadKey($true)
```

You can also use utilities such as SoapUI:



The request returns a block of JSON containing the following:

- `access_token` – your access token.
- `expires_in` – the lifetime in seconds for the token. Once the lifetime has expired, you must request a new access token.
- `token_type` – always `Bearer`.
- `scope` – the scope configured for the client in the web.oauth2 web service; usually `myid.rest.basic`.

You can now use this access token to call the API.

See section *5.1*, *Calling the API from an external system* for more information on using an access token.

3.3.2 Providing a client identifier

MyID captures the IP address and the client identifier of the PC used to carry out the audited operation, and stores this information in the audit trail; see the *Logging the client IP address and identifier* section in the ***Administration Guide*** for more information.

You can provide a client identifier for your grant request by setting a value for the `CLIENT_IDENTIFIER` header in the request.

Note: You can change the name of the header if required; the name of the header is specified in the `MyID:ClientIdentifierHeader` of the `appSettings.json` file of the web.oauth2 web service.

4 End-user authentication

This section provides information on setting up end-user based authentication for the MyID Core API. This allows you to call the MyID Core API using the credentials of a person in the MyID system, using the MyID authentication service to authenticate their credentials, whether security phrases, smart card, FIDO authenticator, or any other authentication method for which MyID is configured.

To set up end-user based authentication, you must carry out the following:

- Configure the web.oauth2 web service with the ID of your client system.

  See section *4.1*, *Configuring web.oauth2 for end-user based authentication*.
- Authenticate your user account and obtain an access token for your system to use.

  See section *4.2*, *Obtaining an end-user based access token using PKCE* or section *4.3*, *Obtaining an end-user based access token using a client secret*.
- Optionally, configure your system to allow refresh tokens.

  See section *4.4*, *Using refresh tokens*.
- Revoke access tokens when they are no longer required.

  See section *4.5*, *Revoking access tokens*.

For more information on end-user authentication, see the *Authorization Code* section of the *OAuth 2.0 Authorization Framework*:

tools.ietf.org/html/rfc6749#section-1.3.1

4.1 Configuring web.oauth2 for end-user based authentication

The MyID authentication web service is called web.oauth2; you must configure this web service to allow access to the API. For end-user based authentication, you do this by configuring the server for acquisition of an access token, secured either by PKCE or a client secret.

Note: Before you begin, you must decide on a client ID for your external system; for example, `myid.myclient`. This represents your application (for example, website) that intends to make calls to the MyID Core API.

4.1.1 Configuring the authentication service for PKCE

For single-page apps, which run entirely on the client PC, you must secure the request for authentication using PKCE. This ensures that only the caller of the authorization can use the authorization code to request an access token.

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. If this file does not already exist, you must create it in the same folder as the `appsettings.json` file.

2. Edit the file to include the following:

```
{
    "Clients":  [
    {
        "ClientId":  "<my client ID>",
        "ClientName":  "<my client name",
        "AccessTokenLifetime":  <time>,
        "AllowedGrantTypes":  [
            "authorization_code"
        ],
        "RequireClientSecret":  false,
        "RequirePkce":  true,
        "AllowAccessTokensViaBrowser":  true,
        "RequireConsent":  false,
        "AllowedScopes":  [
            "myid.rest.basic"
        ],
        "RedirectUris":  [
            "<callback URL>"
        ]
    }
    ]
}
```

where:

- `<my client ID>` – the client ID you decided on; for example: `myid.mysystem`
- `<my client name>` – an easily readable name for your client system; for example: `My Client System`
- `<time>` – the time (in seconds) that the client credential is valid. The default is `3600` – 1 hour.
- `<callback URL>` – the URL of the web page on your system to which the authorization code will be returned.

For example:

```
{
    "Clients":  [
    {
        "ClientId":  "myid.myclient",
        "ClientName":  "My Client System",
        "AccessTokenLifetime":  3600,
        "AllowedGrantTypes":  [
            "authorization_code"
        ],
        "RequireClientSecret":  false,
        "RequirePkce":  true,
        "AllowAccessTokensViaBrowser":  true,
        "RequireConsent":  false,
        "AllowedScopes":  [
        "myid.rest.basic"
        ],
        "RedirectUris":  [
            "https://myserver/mysystem/callback.asp"
        ]
    }
    ]
}
```

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

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.

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**.

   This ensures that the web service has picked up the changes to the configuration file.
5. Check that the web.oauth2 server 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 7, *Troubleshooting* for information on resolving problems that cause HTTP Error 500.30.

You can now obtain an access token; see section *4.2*, *Obtaining an end-user based access token using PKCE*.

4.1.2 Configuring the authentication service for a client secret

For stateful web sites, 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.

Note: The following instructions assume that you are using a client secret without PKCE. If you want to use both a client secret *and* PKCE, you can set both the `RequireClientSecret` and `RequirePkce` options to `true`, and then combine the requests for an authentication code and access token from section *4.2*, *Obtaining an end-user based access token using PKCE* and section *4.3*, *Obtaining an end-user based access token using a client secret*.

Before you edit the configuration file, create a client secret; see section *3.2.1*, *Creating a shared secret*.

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. If this file does not already exist, you must create it in the same folder as the `appsettings.json` file.

2. Edit the file to include the following:

```
{
    "Clients":[
        {
            "ClientId":"<my client ID>",
            "ClientName":"<my client name",
            "AccessTokenLifetime": <time>,
            "AllowedGrantTypes":[
                "authorization_code"
            ],
            "RequireClientSecret":true,
            "RequirePkce":false,
            "AllowAccessTokensViaBrowser":true,
            "RequireConsent":false,
            "ClientSecrets":[
                {
                    "Value":"<secret>"
                }
            ],
            "AllowedScopes":[
                "myid.rest.basic"
            ],
            "RedirectUris":[
                "<callback URL>"
            ]
        }
    ]
}
```

where:

- `<my client ID>` – the client ID you decided on; for example:

  `myid.mysystem`

- `<my client name>` – an easily readable name for your client system; for example:

  `My Client System`

- `<time>` – the time (in seconds) that the client credential is valid. The default is `3600` – 1 hour.

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

  `kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk=`

- `<callback URL>` – the URL of the web page on your system to which the authorization code will be returned.

For example:

```
{
    "Clients":[
        {
            "ClientId":"myid.myclient",
            "ClientName":"My Client System",
            "AccessTokenLifetime":3600,
            "AllowedGrantTypes":[
                "authorization_code"
            ],
            "RequireClientSecret":true,
            "RequirePkce":false,
            "AllowAccessTokensViaBrowser":true,
            "RequireConsent":false,
            "ClientSecrets":[
                {
"Value":"kv31VP5z/oKS0QMMaIfZ2UrhmQOdgAPpXV/vaF1cymk="
                }
            ],
            "AllowedScopes":[
                "myid.rest.basic"
            ],
            "RedirectUris":[
                "https://myserver/mysystem/callback.asp"
            ]
        }
    ]
}
```

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

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.

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**.

   This ensures that the web service has picked up the changes to the configuration file.
5. Check that the web.oauth2 server 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 *7*, *Troubleshooting* for information on resolving problems that cause HTTP Error 500.30.

You can now obtain an access token; see section *4.3*, *Obtaining an end-user based access token using a client secret*.

4.2 Obtaining an end-user based access token using PKCE

Once you have configured the web.oauth2 web service for PKCE, you can request your user-based authentication code, and use that to obtain an access token that you can use to call the API.

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

4.2.1 Generating a PKCE code verifier and code challenge

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 ("`r`nThe code verifier is: `r`n`r`n$code_verifier")
Write-Output ("`r`nAnd code challenge is:`r`n`r`n$clientSecret" )

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

4.2.2 Requesting an authorization code

You must use the PKCE code challenge to request an authorization code from the MyID authentication service, and provide your user credentials.

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

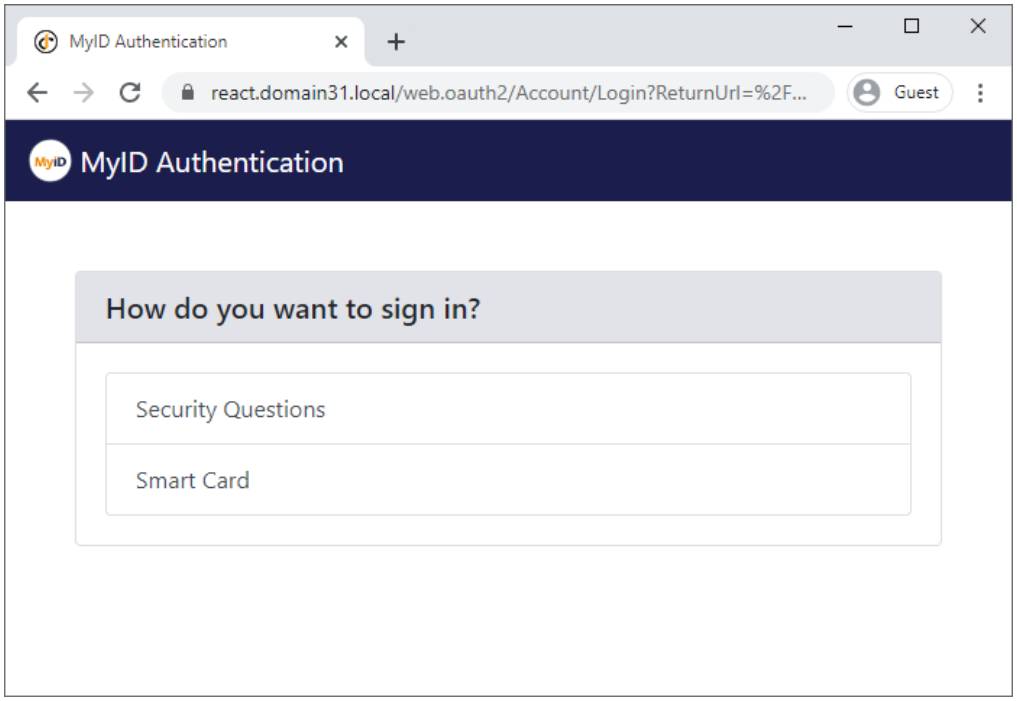
   `https://<server>/web.oauth2/connect/authorize`

   - `client_id` – the ID of your system; for example:

     `myid.myclient`

   - `scope` – set this to `myid.rest.basic`
   - `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` – set this to the code challenge you generated. This is the Base64 URL-encoded SHA-256 hash of the random code verifier you created.
   - `code_challenge_method` – set this to `S256`

   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.



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

   Note: You may need to ensure that the MyID Client Service is running on your PC.

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 access token. You can use the access token repeatedly until it expires, but if you need to request another access token, you must first request another authorization code and go through the user authentication procedure again.

4.2.3 Requesting an access token

Once you have your authorization code, you can request an access token that allows you to call the API.

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

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

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

     `myid.myclient`
   - `code_verifier` – 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.
   - `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 to which the access token will be returned. This must be the same as the URL you specified when you requested the authorization code.

2. Capture the `access_token` that is returned.

   You can now use this access token to call the API.

   See section *5.1*, *Calling the API from an external system* for more information on using an access token.

The request returns a block of JSON containing the following:

- `access_token` – your access token.
- `expires_in` – the lifetime in seconds for the token. Once the lifetime has expired, you must request a new access token.
- `token_type` – always `Bearer`.
- `scope` – the scope configured for the client in the web.oauth2 web service; usually `myid.rest.basic`.

4.3 Obtaining an end-user based access token using a client secret

Once you have configured the web.oauth2 web service for a client secret, you can request your user-based authentication code, and use that to obtain an access token that you can use to call the API.

4.3.1 Requesting an authorization code

You must use the client secret to request an authorization code from the MyID authentication service, and provide your user credentials.

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

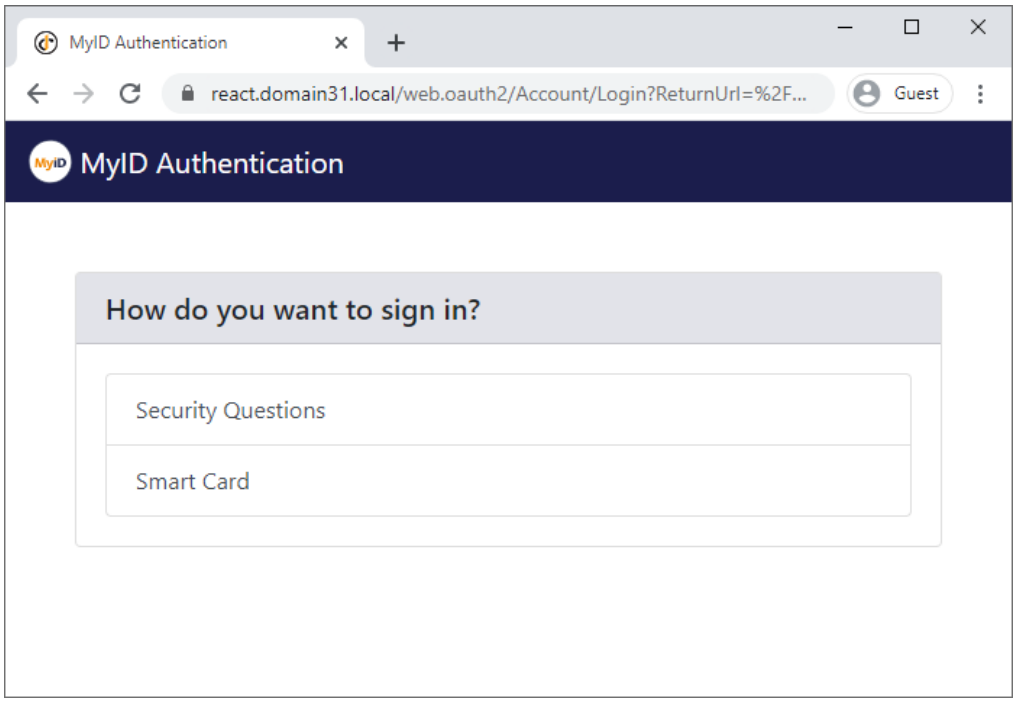
   `https://<server>/web.oauth2/connect/authorize`

   - `client_id` – the ID of your system; for example:

     `myid.myclient`

   - `scope` – set this to `myid.rest.basic`
   - `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.
   - `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.
   - `response_type` – set this to `code`

   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.



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

   Note: You may need to ensure that the MyID Client Service is running on your PC.

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 access token. You can use the access token repeatedly until it expires, but if you need to request another access token, you must first request another authorization code and go through the user authentication procedure again.

4.3.2 Requesting an access token

Once you have your authorization code, you can request an access token that allows you to call the API.

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

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

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

     `myid.myclient`

   - `client_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; see section *3.3.1*, *Requesting an access token* for details.

   - `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 to which the access token will be returned. This must be the same as the URL you specified when you requested the authorization code.

2. Capture the `access_token` that is returned.

   You can now use this access token to call the API.

   See section *5.1*, *Calling the API from an external system* for more information on using an access token.

The request returns a block of JSON containing the following:

- `access_token` – your access token.
- `expires_in` – the lifetime in seconds for the token. Once the lifetime has expired, you must request a new access token.
- `token_type` – always `Bearer`.
- `scope` – the scope configured for the client in the web.oauth2 web service; usually `myid.rest.basic`.

4.4 Using refresh tokens

You can configure the web.oauth2 authentication server to allow you to extend your authenticated session after obtaining the initial access token using a refresh token.

The process is as follows:

1. Configure the authentication server to allow refresh tokens.

   See section *4.4.1*, *Configuring the authentication server*.

2. Call the authentication `/connect/authorize` endpoint with a scope that allows refresh tokens.

   See section *4.4.2*, *Obtaining the authorization code*.

3. Call the authentication `/connect/token` endpoint and receive a refresh token in addition to the access token.

   See section *4.4.3*, *Obtaining the access and refresh tokens*.

4. If the access token has expired, or is close to expiry, call the authentication `/connect/token` endpoint with the refresh token to obtain a fresh access token and refresh token without having to re-authenticate.

   See section *4.4.4*, *Using the refresh token to obtain a new access token*.

5. Repeat the process of obtaining fresh access tokens and refresh tokens as often as required. You can use a refresh token up until its expiry (by default, after two hours) so if you are continually making calls to the API, and can request a fresh access token and refresh token every two hours, you do not need to authenticate until you hit the limit (by default, six days).

This is the same feature of the authentication server that is used for timeouts and re-authentication for the MyID Operator Client; for information about how this system works, see the *Timeouts and re-authentication* section in the ***MyID Operator Client*** guide.

4.4.1 Configuring the authentication server

You can configure the authentication server for end-user authentication using either PKCE or client secrets; see section *4.1.1*, *Configuring the authentication service for PKCE* or section *4.1.2*, *Configuring the authentication service for a client secret*.

To allow the use of refresh tokens, you must make the following additional configuration changes:

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. If this file does not already exist, you must create it in the same folder as the `appsettings.json` file.

2. In the client section for your API client, add the following settings:

   - `AllowOfflineAccess` – set to `true` to allow refresh tokens.
   - `SlidingRefreshTokenLifetime` – the number of seconds within which you can extend the authentication. The default is `7200` (two hours).
   - `AbsoluteRefreshTokenLifetime` – the number of seconds after which you must re-authenticate, even if you have been continually extending the authentication. The default is `518400` (six days).

   For example:

   ```
   "Clients":  [
     {
     "ClientId":  "myid.mysystem",
     "ClientName":  "My External System",
     "AllowOfflineAccess":  true,
     "SlidingRefreshTokenLifetime":  7200,
     "AbsoluteRefreshTokenLifetime":  518400,
     ...
   ```

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**.

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

4.4.2 Obtaining the authorization code

You can call the `/connect/authorize` endpoint to obtain an authorization code using either PKCE or a client secret; see section *4.2.2*, *Requesting an authorization code* (for PKCE) or section *4.3.1*, *Requesting an authorization code* (for client secrets).

When you request the authorization code, instead of requesting a scope of `myid.rest.basic`, request a scope of:

```
myid.rest.basic offline_access
```

This means that when you request an access token using the returned authorization code, it will additionally provide a refresh token.

4.4.3 Obtaining the access and refresh tokens

You can call the `/connect/token` endpoint to obtain an access token using either PKCE or a client secret; see section *4.2.3*, *Requesting an access token* (for PKCE) or section *4.3.2*, *Requesting an access token* (for client secrets).

If you have configured the authentication server to allow it, and added the `offline_access` scope to the request for the authorization code, the request returns a block of JSON containing the following:

- `access_token` – your access token.
- `expires_in` – the lifetime in seconds for the access token. Determined by the `AccessTokenLifetime` setting in the web.oauth2 application settings file.
- `token_type` – always `Bearer`.
- `refresh_token` – your refresh token, which will be valid for the number of seconds determined by the `SlidingRefreshTokenLifetime` setting in the web.oauth2 application settings file.
- `scope` – `myid.rest.basic offline_access`.

4.4.4 Using the refresh token to obtain a new access token

If the access token has expired, or is about to expire (which you can determine from the `expires_in` node of the returned JSON), you can use your refresh token to obtain a fresh access token.

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

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

   - `client_id` – the ID of your system; for example:

     `myid.myclient`

   - `grant_type` – set this to `refresh_token`
   - `refresh_token` – set this to refresh token you obtained previously.

2. Capture the fresh `access_token` and `refresh_token` that are returned in the block of JSON.

   You can now use this access token to call the API, and can use the new refresh token to obtain further access tokens as necessary.

Note: If the old access token has not yet expired, you can continue to use it; requesting a fresh access token does not invalidate the previous one. However, you can use a refresh token only once.

If your access token has expired and your refresh token has expired, or if you have exceeded the limit since the original authorization code was requested (as determined by the `AbsoluteRefreshTokenLifetime` setting in the web.oauth2 application settings file) you must call `/connect/authorize` again to re-authenticate.

If the refresh token has expired, or if the `AbsoluteRefreshTokenLifetime` limit has been exceeded, a response of `invalid_grant` is returned.

4.5 Revoking access tokens

You can revoke an access token for the web.oauth2 authentication server; the server supports the OAuth2/OpenID Connect revocation endpoint.

Where the token contains a scope that allows accessing the MyID Core API, the revocation endpoint updates the MyID database to ensure that the access token can no longer be used.

If you are using refresh tokens , these are also invalidated.

Note: Revoking a token that does not include a scope that relates to the MyID database (for example, if it does not have `myid.rest.basic` scope because the token is used for products outside of MyID) invalidates any refresh tokens, but the access token remains valid until expiry. This is because an access token is valid until expiry, *unless* there is a back-end system that can be updated to indicate that access token is no longer valid.

To call the revocation endpoint, post the following information to the MyID revocation URL, formatted according to the RFC for OAuth 2.0 Token Revocation (RFC 7009):

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

- `client_id` – the ID of your system; for example:

  `myid.myclient`

- `token` – set this to the access token.

- `token_type_hint` – set this to:

  `access_token`

5 Calling the API

You must configure your server for the appropriate method of authentication.

- To configure the server and obtain an access token for server-to-server authentication, see section *3*, *Server-to-server authentication*.
- To configure the server and obtain an access token for user authentication, see section *4*, *End-user authentication*.

Once you have obtained an access token, you can call the API from an external system; see section *5.1*, *Calling the API from an external system*.

You can also call the API from within the Swagger-based documentation; see section *5.2*, *Calling the API from the documentation*.

5.1 Calling the API from an external system

Once you have obtained an access token, you can call the API.

Present the access token in an `Authorization` header with a type of `Bearer` (as detailed in the `token_type` option in the returned JSON containing your access token).

Important: You must keep track of the lifetime of the token, and request a new token when the current token has expired. Avoid obtaining a new access token if the previous token is still valid, as this generates unnecessary records in the `Logons` table in the MyID database, and may impact your system performance.

For example, to view the details of a particular person, you use the following:

```
GET /api/People/{id}
```

This method requires a parameter `dirInfo` and the ID of the person – this example uses the following ID:

```
53F2A29B-376B-4600-867C-2E5BD95AE222
```

See the API documentation for the particular requirements for each method.

For example, using cURL:

```
curl -k -i -X GET "https://myserver.example.com/rest.core/api/People/53F2A29B-376B-4600-
867C-2E5BD95AE222" -d "dirInfo=true" -H "Authorization: Bearer <token string>"
```

or using PowerShell:

```
# Access token for the API
$token = "<token string>"

# Set the body of the request. The parameters depend on the method being used;
# see the Swagger-based API documentation for details.
$body = @{dirInfo='true'
    }

# Set the headers for the request
$header = @{Authorization="Bearer $token"
    }

# Call the method. This example obtains the information about a particular
```

```
# person. The user configured for the API must have access to this feature
# through their role assignment, and must have scope that allows them to
# view the details of the specified person.
Invoke-WebRequest -Method GET -Uri
'https://myserver.example.com/rest.core/api/People/53F2A29B-376B-4600-867C-2E5BD95AE222' -
body $body -Headers $header | Select-Object -Expand Content

# Wait for a keypress
Write-Host "`r`nPress any key to continue..." -ForegroundColor Yellow

[void][System.Console]::ReadKey($true)
```

5.2 Calling the API from the documentation

The Swagger-based documentation for the API allows you to try the methods from within the documentation itself, once you have configured the server for the appropriate method of authentication.

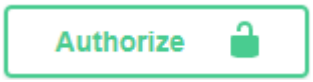
5.2.1 Calling the API from the documentation using server-to-server authentication

Once you have configured the web server for server-to-server authentication, you can also use the client ID and shared secret to authenticate to the server to access the API features from the Swagger-based API documentation:

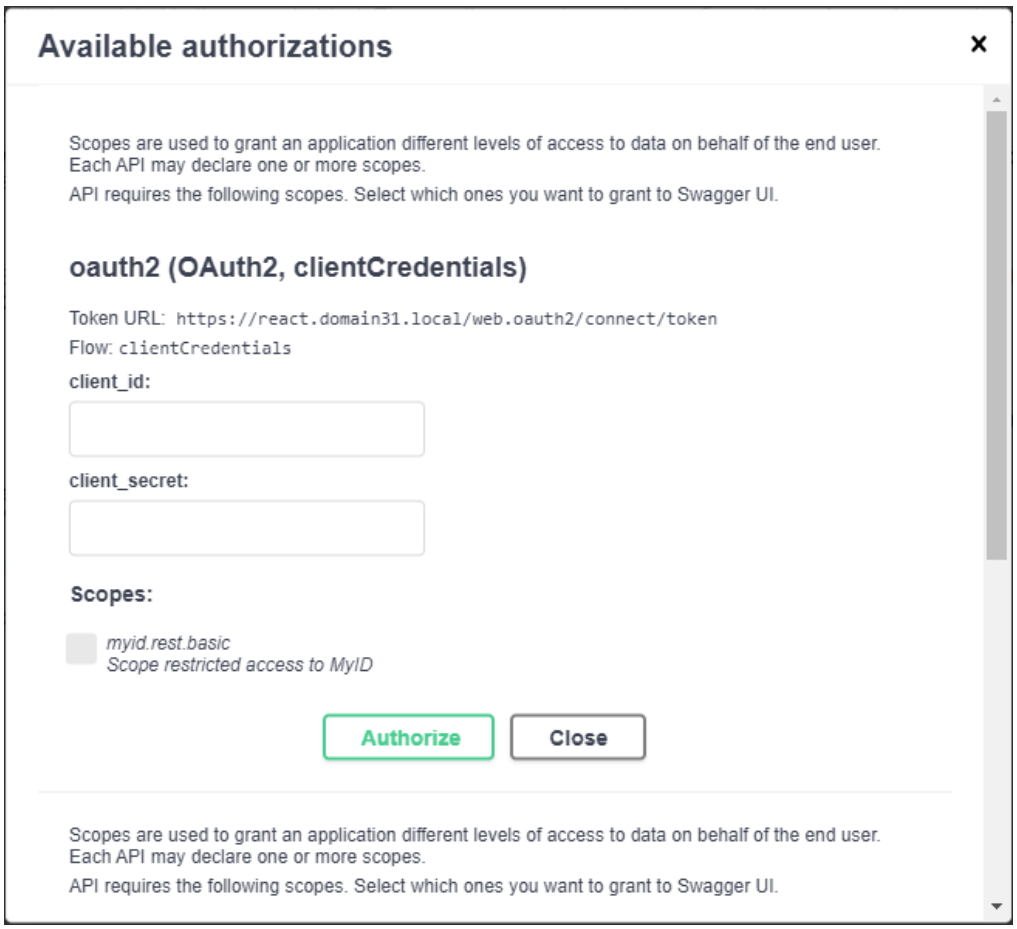
1. Open the API documentation in a browser.

   See section *2.1*, *Accessing the API documentation* for details.

2. Click **Authorize**.

   The Available authorizations screen appears.



3. In the **oauth2 (OAuth2, clientCredentials)** section, enter the following:
   - **client_id** – your client ID; for example, `myid.mysystem`.
   - **client_secret** – your client secret; for example:

     `82564d6e-c4a6-4f64-a6d4-cac43781c67c`

     Note: Do not use the combined client ID and Base64 version of the client secret.
   - **Scopes** – select the **myid.rest.basic** option.
4. Click **Authorize**.
5. Click **Close**.

5.2.2 Calling the API from the documentation using end-user authentication

You can use end-user authentication to access the API from within the documentation.

1. Edit the `appsettings.json` file for the web.oauth2 web service to include the Swagger callback URL.
   a. Open the `appsettings.json` file in a text editor.

      By default, this is:

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

      Note: If you have an `appsettings.Production.json` file, use that instead. This is the override configuration file for the `appsettings.json` file for the web service.

b. In the `Clients` section, under the `myid.operatorclient` settings, add the following to the `RedirectUris` array:

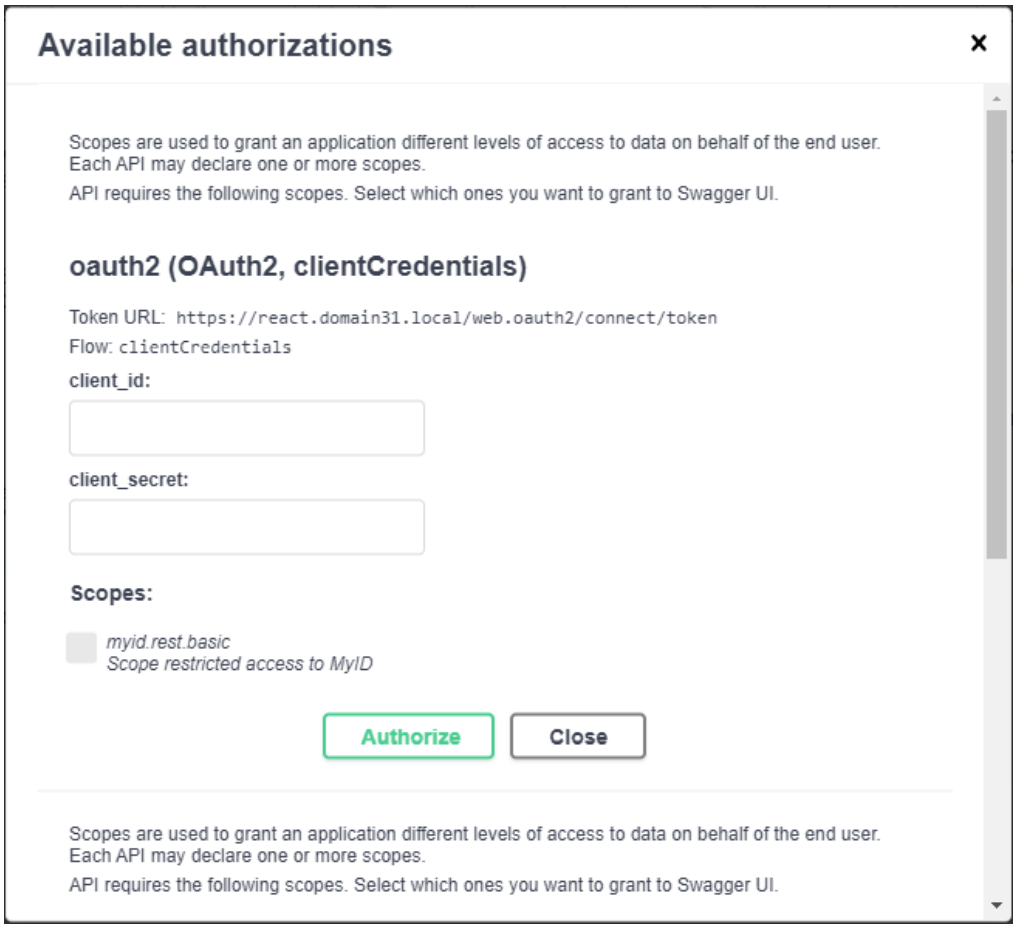
```
https://<server>/rest.core/swagger/oauth2-redirect.html
```

where `<server>` is the name of your MyID authentication. For example:

```
"RedirectUris": [
"https://react.domain31.local/MyID/OperatorClient",
"https://react.domain31.local/rest.core/swagger/oauth2-redirect.html"
]
```

2. Click **Authorize**.

3. The Available authorizations screen appears.



4. Scroll to the **oauth2 (OAuth2, authorizationCode)** section, and enter the following:

- **client_id** – enter `myid.operatorclient`.

  If you have set up your own system to use end-user authentication, you can use this client ID instead.

- **Scopes** – select the **myid.rest.basic** option.

5. Click **Authorize**.
6. Complete the authentication using your method of choice; for example, security questions or smart card.

   Note: You may need to ensure that the MyID Client Service is running on your PC.
7. Click **Close**.

6 Operation extension

You can extend the authorization for your API calls to call additional operations through the MyID Client Service; this allows you to carry out some operations (for example, resetting a device PIN) by launching MyID Desktop;. you can use this feature to carry out actions that are not yet supported by the MyID Core API. You can also carry out operations provided by the MyID Client Service; for example, using the Select Security Device dialog to select a device.

You can carry out the following:

- Using the MyID Client Service API to launch an operation.

  See section *6.1*, *Obtaining an operation extension token*.

- Using the MyID Client Service API to open the Select Security Device dialog.

  section *6.2*, *Obtaining an extension token for Select Security Device*.

The MyID Client Service API documentation is available in the MyID Integration Toolkit, available on request.

6.1 Obtaining an operation extension token

Before you can call the MyID Client Service API to launch MyID Desktop, you must obtain an operation extension token for this particular operation on this particular object (for example, device, request, or person) – this is a short-lived authorization code for a single use.

When you call the MyID Core API, the block of data returned contains a series of links. If a link has a `cat` of `myid.mcs`, this means you can carry out this operation using the MyID Client Service.

For example:

```
{
"op":"297",
"cat":"myid.mcs",
"desc":"Reset Card PIN",
"verb":"",
"auth":"client_id=myid.mcs&grant_
type=operation&op=297&scope=myid.operation&deviceId=<DEVICE ID>&token={token}",
"clientData":["DSK"]
}
```

You can use this information to obtain an operation extension token, then use this token to call the MyID Client Service API to launch the workflow.

To request the operation extension:

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

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

   - `client_id` – set this to `myid.mcs`
   - `grant_type` – set this to `operation`
   - `op` – set this to the ID of the operation you want to carry out.

     For example:

     - Operation ID `297` is **Reset PIN**.
     - Operation ID `5007` is **Assisted Activation**.

     You can obtain the operation ID from the link data.
   - `scope` – set this to `myid.operation`
   - `deviceId`, `personId`, or `requestId` – provide the ID of a single device, person, or request. You can obtain the ID from the link data.
   - `token` – set this to your existing authorization token.

     See section *3.3*, *Obtaining a server-to-server access token*, section *4.2*, *Obtaining an end-user based access token using PKCE*, or section *4.3*, *Obtaining an end-user based access token using a client secret* for details.
2. Capture the access token that is returned.

   You can now use this access token in the `Token` argument of the `StartWithToken` method of the MyID Client Service API to launch MyID Desktop for the specified operation and target.

6.2 Obtaining an extension token for Select Security Device

Before you can call the MyID Client Service API to open the Select Security Device dialog authenticated with the logged-on operator, you must obtain an extension token for this particular operation – this is a short-lived authorization code for a single use.

Note: This authenticated mode provides user images and full names on the smart card selection screen based on the scope and administration groups of the logged-on user. If you do not need to display this additional detail, you can call the `SelectCard` method of the MyID Client Service API without the `Token` parameter; in this case, you do not need to obtain an extension token.

To use authenticated mode, you must ensure that the MyID web.oauth2 server is configured to allow a scope of `myid.devicepicker`. Check the `appsettings.json` file (by default, in the `C:\Program Files\Intercede\MyID\web.oauth2\` folder) for the following:

- In the `Scopes` array, a scope called `myid.devicepicker` exists, with the following `UserClaims`:
  - `deviceDetectContext`
  - `op`
  - `myidSessionId`

- In the `ApiResources` array, for the resource with name `myid.mws`, the scope `myid.devicepicker` is in the `Scopes` array.
- In the `Clients` array, for the client with ID `myid.mcs`, the scope `myid.devicepicker` is in the `AllowedScopes` array.

To obtain the extension token:

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

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

   - `client_id` – set this to `myid.mcs`
   - `grant_type` – set this to `operation`
   - `op` – set this to the ID of the **Read Card (Authenticated)** operation. This is `100221`.
   - `scope` – set this to `myid.devicepicker`
   - `token` – set this to your existing authorization token.

     See section *3.3*, *Obtaining a server-to-server access token*, section *4.2*, *Obtaining an end-user based access token using PKCE*, or section *4.3*, *Obtaining an end-user based access token using a client secret* for details.
   - `deviceDetectContext` – reserved for future use. Leave as an empty query parameter.
2. Capture the access token that is returned.

   You can now use this access token in the `Token` argument of the `SelectCard` method of the MyID Client Service API to launch the Select Security Device dialog authenticated with the logged-on operator.

7 Troubleshooting

If you experience problems when attempting to use the MyID Core API:

- Check that you have set up the server-to-server authentication correctly.

  See section *3*, *Server-to-server authentication*.

- For PKCE-based end-user authentication, check that you have set up PKCE correctly.

  See section *4.2.1*, *Generating a PKCE code verifier and code challenge*, and see the PKCE standard for details of requirements for the code verifier and code challenge:

  tools.ietf.org/html/rfc7636

  Make sure you are using Base64 URL encoding to create the code challenge, and not simply Base64 encoding.

- Review the documentation to ensure that you are using the API calls correctly.

  See section *2.1*, *Accessing the API documentation*.

- Check that you have configured access to the API for the features you want to use.

  See section *2.2*, *Accessing the API features*.

- If you cannot see the people, requests, devices and so on that you expect, check that the operator account used to access the API has the correct scope.

  See section *2.2.1*, *Scope*.

- 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.

- 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
  ```

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.

- If you see an error message, look up the error code.

  See the ***Error Code Reference*** guide for a list of errors codes, their causes, and potential solutions.

- Enable logging on the rest.core and web.oauth2 web services.

  See the *MyID REST and authentication web services* section in the ***Configuring Logging*** guide.