

MY ONTARIO ACCOUNT
FOR HEALTH

Integration Guide

VERSION 1.3.1 | October 2026

Ontario Health



Copyright Notice

Copyright© 2026, Ontario Health (Digital Services)

All rights reserved

No part of this document may be reproduced in any form, including photocopying or transmission electronically to any computer, without prior written consent of Ontario Health (Digital Services). The information contained in this document is proprietary to Ontario Health (Digital Services) and may not be used or disclosed except as expressly authorized in writing by Ontario Health (Digital Services).

Trademarks

Other product names mentioned in this document may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.

Approval History

APPROVER(S)	TITLE/DEPARTMENT	APPROVAL DATE
Digital Health Interoperability Standards Committee	Ontario Health	Oct 28, 2026
AODA Remediation	Ontario Health	Oct 28, 2026

Revision History

Version No.	DATE	SUMMARY OF CHANGE	CHANGED BY
1.3	June 16, 2026	Initial release	Ontario Health (Digital Services)
1.3.1	September 1, 2026	Formatting and style	Ontario Health (Digital Services)

Table of Contents

Introduction 5
About this Document 5
Intended Audience 5
Integration Pattern 6
Overview 6
Open ID Connect (OIDC) 9
OIDC Metadata 9
Authorize Client “authorization_endpoint” 9
Client Callback and Authorization Code from the Authorization Server 11
Access Token Request “token_endpoint” 11
UserInfo Request "userinfo_endpoint" 15
User Logout Request 16
Smart on FHIR (SoF) Standalone Flow 17
Overview 17
Authorize Client 17
Client Callback and Authorization Code from the Authorization Server 18
Access Token Request 19
Refresh Token Request 20
Requesting Health Records 21
Token Revocation 22
Overview 22
Access / Allow-listing 23
Overview 23
Appendix 24
My Ontario Account for Health Account Behaviors 24
Session Behaviors: Definitions 24
Glossary 26
Reference Material 27
Sample OIDC Metadata 28

Introduction

About this Document

My Ontario Account for Health (MOAH) uses Ontario.ca Login credentials combined with identity verification with a valid Ontario Health Card. This involves comparing a photo of your health card with a photo of you and validating the information on the health card. MOAH allows access to various health records and services provided by the Ontario government, such as viewing lab results and publicly funded dispensed medications, through digital health tools. For more information about how Ontario Health collects, uses and discloses PHI as a Prescribed Organization in respect of Digital Health Identifier activities, please visit Digital Health Identifier For additional information, please refer to Reference Material.

This document will describe the integration requirements for a third party to consume a My Ontario Account for Health including both verified identity information about an individual and/or a user's health records. The integration pattern uses a combination of OpenID Connect (OIDC) and Smart on FHIR (SoF).

Intended Audience

The intended audience for this document are individuals responsible for understanding the technical details on how information access is authorized and accessed or are involved in directly implementing a solution using this integration.

Integration Pattern

Overview

The Client follows an OAuth 2.0 Authorization pattern. First, it requests access to resources from the Authorization Server (1). After the end-user interacts with the Authorization Server, the Client receives a callback with an authorization grant (2). The Client can redeem the grant for an access token (2), for use directly with the resource server (3) to receive the protected resource owned by the end-user (4). Details about the resource(s) (3a) may be known ahead of time by the client, discovered through a well-known endpoint, or communicated in the grant process: such as through the patient parameter in SoF token responses.

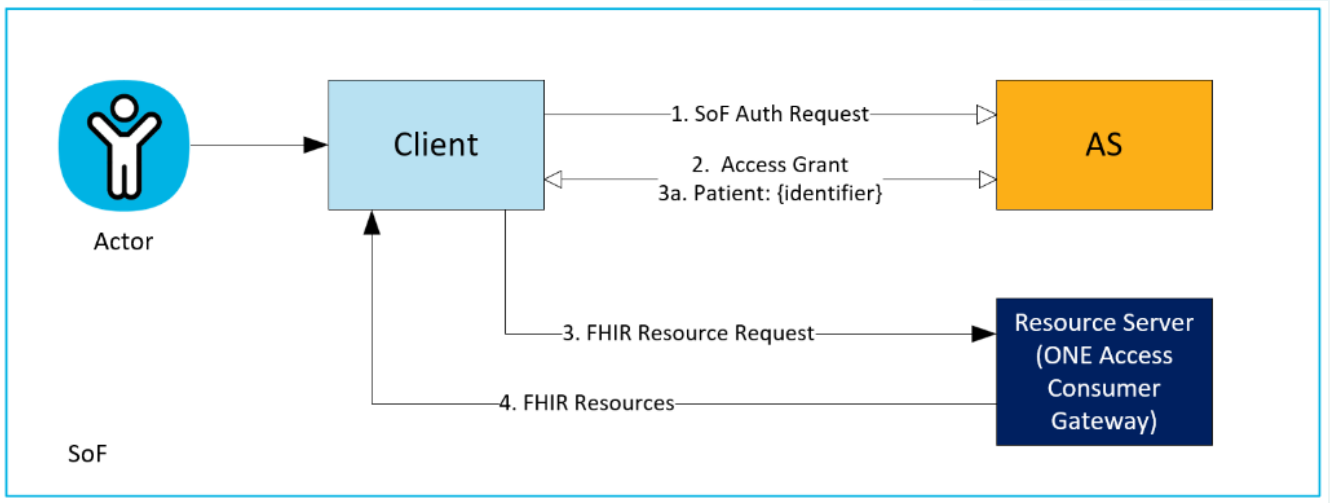


Figure 1: Overview of Client, Authorization Server and Consumer Gateway

In the sections below there will be several values that will differ depending on the specific integration environment.

VALUE	DESCRIPTION
authorization_server_base_url}	This will reference the base URL of the Authorization Server that will be used to grant access to specific user resources.
{health_record_uri}	This will reference the base URL of the Consumer Gateway Application that the client will request health records from
{client_redirect_url}	The URL that the client application will provide that the user will be returned to after completing an authorization request. This value is pre-authorized and must be provided to the authorization server ahead of time. No wildcards are allowed.
{client_id}	The OAuth “client id” provided to the client for integration purposes to identify them in all requests.

During an authorization transaction, the Client may use the following endpoints at the Authorization Server:

1) Well known Endpoint: A back-channel request to discover the Authorization endpoint, Token endpoint, and UserInfo endpoint
2) Authorization Endpoint: An end-user-facing endpoint that users will be redirected to complete an authorization request
3) Token Endpoint: A back-channel request to receive access tokens after the request is completed (also used to refresh an existing access token.
4) UserInfo Endpoint: A back-channel request to receive user details about a patient using an access token
5) Logout Endpoint: A front channel request to log the current user out (only supported by OIDC)

Client Authentication

Regardless of the OAuth profile used, the client must authenticate themselves during the back-channel request to acquire an access token, this is done by building a JSON web token (JWT), signed by the client when requesting the access token. Clients will be provided a "clientid" in advance of the integration, and clients will provide a front-channel redirect URI that will be whitelisted by the authorization server. Clients must also provide the public key for the private key they will sign JWTs with, either through a JWKS endpoint, or by providing the raw public key.

It is strongly suggested to make use of a library to handle the signing and verification of JWTs.

To use JWT authentication, the Client must build a JWT containing a specific set of claims and signed with the private key that can be verified by the Authorization Server using the public key set provided during registration. For illustration purposes, see sample below:

```
{
 "iss": "{client_id}",
 "sub": "{client_id}",
 "aud": "https://{authorization_server_base_url}/transaction/token",
 "exp": 1619831987,       // JWT expiry - should be a future UNIX timestamp
 "iat": 1619826987,       // JWT valid from - should be past UNIX timestamp
 "jti": "yrqjsBgSzqFOXv9TwDIOsKFKlFEA7aPGa7lm" // must be unique per request
}
```

This JWT should be signed using the confidentially held private key and would result in the following:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.Qp7AjM0UPIXdSTaRK5jIdTcj1cosLWPXbt0CXqR3FxNvMW5kaj
Ccxzg0DVQ9k8yE4G5VHktvbqOVKEVsSg8JjVmjs_Y4rxiWTS6uOG7r0_xGuk3FEJx_iUsAKHvVbjzOzFhfo1
XK_GHmsRxV9bYOSTlBfdO0Q49qZtQ

See specific OAuth profiles below on how to use this JWT to authenticate during a user authorization request.

Client Registration Process

OAuth Clients will be onboarded through a static registration process. The Client must provide their public key details, scopes desired for integration, and callback uris (including logout and login) to the Ontario Health team for registration. Once registered, the client will be given a client identifier, and they must undergo client testing first through the integration environment and final sign off, before being approved for production.

Errors Responses

In all cases, there are two types of client-facing errors that will be returned. Both authentication errors as well as invalid parameters will respond with a 400 HTTP status code with the following structure:

```
{
 "error": "invalid_request”,
 "error_description": "Error description"
}
```

Open ID Connect (OIDC)

OIDC requests can be combined with SoF through the use of the `openid` scope. The integration pattern remains the same as SoF, there is no requirement to request other SoF scopes. All of the endpoints described in this section can be discovered from the authorization server's OIDC metadata endpoint.

Note: while a dynamic registration endpoint may be discoverable at the metadata endpoint, clients are not authorized to use it and must use static registration as per the Client Registration Process.

OIDC Metadata

The relying party can request the OIDC metadata endpoint at the standard issuer location. Refer to the Appendix for an illustration.

Authorize Client “authorization_endpoint”

The end user should be redirected to the following endpoint to begin an authorization request.

For illustration purposes, see sample below:

https://{authorization_server_base_url}/oauth2/authorize?&client_id={client_id}&state=1648810292&redirect_uri={client_redirect_url}&response_type=code&scope=openid&code_challenge_method=S256&code_challenge=nxAvddK_1ECz_2oDLMevOnJbjPEBeNSEbdx96iuqVJg&nonce=RGfvbrKJ9bhmflKdqZcxWA

PARAMETER	REQUIRED	DESCRIPTION
client_id	YES	OAuth 2.0 Client ID.
redirect_uri	YES	One of the registered redirect URIs.
state	YES	Opaque string to associate the authentication request during the callback
response_type	YES	Specifies that the application is requesting an authorization code grant.
scope	YES	This will include a space delimited list of scopes. Limited to the openid and should include scopes representing the scope of access required as per sehttps://openid.net/specs/openid-connect-core-1_0.html#ScopeClaims. Within the solution these are limited to the following read only scopes: patient/Patient.read user/MedicationDispsense.read user/DiagnosticReport.read
nonce	YES	A unique opaque string required specifically for OIDC requests. Nonce is generated by the client prior to sending an OIDC request. It is used to prevent replay attacks and also to verify the id_token received from the Authorization Server.
code_challenge	YES	The code challenge is a BASE64-URL-encoded string of the SHA256 hash of the code verifier
code_challenge_method	YES	The hash function used to generate code challenges from the code verifier. Must be S256.
ui_locales	YES	BCP47 language tag value. Only “en” and “fr” are supported. This will set the initial language preference for the user within the authorization flow.
login	NO	Indicates that the authorization request should be treated as an existing user and will redirect the user directly to MOA. Its value should be “true” when used.

Client Callback and Authorization Code from the Authorization Server

When the user approves the authorization request, the service redirects the user-agent to the application redirect URI, which was specified during the client registration, along with an Authorization Code.

Based on the redirect URI passed in the authorize step above, here's a sample of a successful response:

https://{client_redirect_url}?code=361e4986-56b2-446f-9718-b3794305f388&state=1654029277

If an error occurred or the end-user did not grant the client access, the Authorization Server will redirect back to redirect_uri provided with an error. Here's a sample of an error:

https://{client_redirect_url}?error=access_denied&state=1654029277&error_description=Access+denied+by+resource+owner+or+authorization+server

Access Token Request “token_endpoint”

The client application requests an access token from the API by passing the authorization code that was returned during the callback along with client authentication details (this flow matches the SoF token request exactly).

Request

For illustration purposes, see sample below:

POST /oauth2/token
curl --location --request POST 'https:// {authorization_server_base_url}/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'redirect_uri={client_redirect_url}' \
--data-urlencode 'code=df4ba24c-ebfb-43b8-90d9-64d90f7dd647' \
--data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \
--data-urlencode
'client_assertion=eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJjbGllbnRfaWQiLCJhdWQiOiJ1cmx0aGluZ3MiLCJpc3MiOiJjbGllbnRfaWQiLCJleHAiOjE2NTQzMjE4MTMsImlhdCI6MTY1NDMwMDIxMywianRpIjoiMDMyZTcyNDItOWFhYi00ZDkxLWI2NjYtZDcyMjgwMTU0MjkxIn0.ht8P5FeTze0-2Y-ISsmxTUaTF246weLxjvbfX4Zb3cfrpPmdW2trg6bGgfq8lBDtkOyPzmLFieUxkNtfZDpqYV6UgRtaVcqerCysEVLHHIaxa5mB7tfUQtBFzIidVDoPfvAEOmMowc6F7whH1IuOmDTp6szfm_uYEHZzLNEERm_bXr0R6uN-ig-l6bgVNsTvswtvRx7AtlvOb0B62FH1Kp4mLc7W8JoqMkowfV-UVXKoFFtIOQj6d9omwEHljWPY9BR_YtUVfkGzxTk1S3Ieax0tQT4Ke0ed9kPTviLKAD04D7lrRu2gpPAq4y8S1UEVshddQmDZEt7WlgmgDkK5pw'

PARAMETER	REQUIRED	DESCRIPTION
grant_type	YES	Must be 'authorization_code'
redirect_uri	YES	The redirect URI that was sent with the authorization request in 2.2.1
code	YES	Actual authorization code value that was received in 2.2.2
client_assertion_type	YES	Must be set to urn:ietf:params:oauth:client-assertion-type:jwt-bearer
client_assertion	YES	Signed JWT (see 2.1.1 Client Authentication)
code_verifier	YES	The code challenge from 2.2.1 is the BASE64-URL-encoded string of the SHA256 hash of the code verifier.

Response

The endpoint will return the access token to be used against the resource server to access protected resources along with its expiry, a refresh token, what scopes were granted as part of the authorization request, and an id_token representing the current user.

For illustration purposes, see sample below:

{
  "access_token": "eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.
lS6bq4GpJi9xApPLYWqRxo83z2Y5a0S6icoNfcwDzrGqHoZaO4wcAw9fkgnCPSgW7Ful7rygZCvsnPDyDCb
nW0iJlCNJ1BHgv _QDMRCQ",
  "expires_in": 3600,
  "token_type": "Bearer",
  "refresh_token": "e0dd0169-36a9-4cf4-b30a-3b78c6162bc3",
  "scope": "read"
 "id_token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ._E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoSK5
hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4XUVrWOLrLl0nx7RkKU8NXNH
q-rvKMzqg"
}

RESPONSE PARAMETER	DESCRIPTION
access_token	The OAuth 2.0 token used against APIs to retrieve protected resources
refresh_token	The refresh token to be used to get an updated access token after the initial one has expired
token_type	This will always be “Bearer”
expires_in	Seconds before the token will expire
scope	The resource server specific scopes that were granted as part of the authorization request for the user
id_token	An encoded JWT that ascertains the user has been authenticated with an identity provider. The claims within the token serve as declarations about the user.

ID Token Structure

The ID token returned from the authorization server will contain the minimum data required by the OIDC specification which includes the subject (sub) only. Additional information about the user should be pulled from the user info endpoint. Note that the “sub” field is considered personal health information (PHI) and should be treated as such. For illustration purposes, see sample below:

```
{
  "sub": "c1c6192c-7f55-4224-8dab-14e2c678441e",
  "aud": "{client_id}",
  "iss": "https://{authorization_server_base_url}",
  "exp": 1706125728,
  "nonce": "0e74b4f4-d8a9-54c3-af98-592c1a451c49",
  "iat": 1706125428
}
```

Refresh Token Request

The access tokens provided by the token endpoint will expire quickly. In order to receive new access tokens, a subsequent request to this endpoint should be made using the refresh token. For illustration purposes, see sample below:

POST /oauth2/token

curl --request POST ' https://{authorization_server_base_url}/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'redirect_uri={client_redirect_url}' \
--data-urlencode 'refresh_token=df4ba24c-ebfb-43b8-90d9-64d90f7dd647' \
--data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \
--data-urlencode 'client_assertion=eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.
UVXKoFFtIOQj6d9omwEHljWPY9BR_YtUVfkGzxTk1S3Ieax0tQT4Ke0ed9kPTviLKAD04D7lrRu2gpPAq4y8
S1UEVshddQmDZEt7WlgmgDkK5pw'

PARAMETER	REQUIRED	DESCRIPTION
grant_type	YES	Must be ‘refresh_token
redirect_uri	YES	The redirect URI that was sent with the authorization request in 2.2.1
refresh_token	YES	The refresh token received from a previous call to the token endpoint
client_assertion	YES	Signed JWT (see 2.1.1 Client Authentication)
client_assertion_type	YES	Must be set to urn:ietf:params:oauth:client-assertion-type:jwt-bearer

Response

The response will return a new access token that can be used.

RESPONSE PARAMETER	DESCRIPTION
access_token	The OAuth 2.0 token used against APIs to retrieve protected resources
refresh_token	The refresh token to be used to get an updated access token after the initial one has expired
token_type	bearer
expires_in	Seconds before the token will expire
scope	The resource server specific scopes that were granted as part of the authorization request for the user

UserInfo Request "userinfo_endpoint"

After an access token has been received, the relying party can request the users information that will expand on the information provided in the id_token. The UserInfo endpoint can be known by the client ahead of time or dynamically pulled from the metadata endpoint at the authorization server. At this time the user info endpoint will return the subject identifier alone but will not contain any user claims. For illustration purposes, see sample request and response below:

Request

GET https://{userinfo_base_url}/api/userinfo
curl --location --request GET https://{userinfo_base_url}/api/userinfo \
--header 'Authorization: Bearer {access_token}'

Response

```
{
  "sub": "3a7c0652-7f28-480b-8801-d92ef7580f17"
}
```

Note that the “sub” field is considered personal health information (PHI) and should be treated as such.

User Logout Request

The relying party can also request that the current users session be ended immediately. To do so a request is made to the “end_session_endpoint”. For illustration purposes, see sample of request and response below:

Request

GET https://{userinfo_base_url}/account/logout

PARAMETER	REQUIRED	DESCRIPTION
id_token_hint	NO	The ID token that was provided to the relying party during the initial authorization request.
post_logout_redirect_uri	YES	One of the registered logout redirect URIs that the user will be sent back to after a successful logout.

Response

The service will respond with a 302 redirect where the user will be logged out of additional systems before returning the user back to the relying party.

Smart on FHIR (SoF) Standalone Flow

Overview

SoF is an extension of the OAuth 2.0 framework. The SMART App Launch Framework is supported along with the v1.0 scope set. Clients will only be allowed to request the minimum set of scopes required to fulfill their needs. That minimum set must be negotiated during the registration process. Only the patient launch context is supported (launch/patient).

Authorize Client

To begin, the user agent should be redirected to the authorization endpoint at the authorization server to begin an authorization request. For illustration purposes, see sample below:

https://{authorization_server_base_url}/oauth2/authorize?&client_id=client_id&state=1648810292&redirect_uri={Client_redirect_url}&response_type=code&scope=openid_launch/patient_patient/patient.read&code_challenge_method=S256&code_challenge=nxAvddK_1ECz_2oDLMevOnJbjPEBeNSEbdx96iuqVJg&aud={health_record_uri}

PARAMETER	REQUIRED	DESCRIPTION
client_id	YES	OAuth 2.0 Client ID
redirect_uri	YES	One of the registered redirect URIs.
state	YES	Opaque string to associate the authentication request during the callback
response_type	YES	Specifies that the application is requesting an authorization code grant.
scope	YES	SoF scope requested by the client. The scopes should represent the minimum scope of access required for the clients purposes (do not request patient/*.*. access when patient/Observation.read is enough). This must always include at a minimum the scopes “openid” and “launch/patient”
aud	YES	the base uri that the client will request health records from
code_challenge	YES	The code challenge is a BASE64-URL-encoded string of the SHA256 hash of the code verifier

PARAMETER	REQUIRED	DESCRIPTION
code_challenge_method	YES	The hash function used to generate code challenges from the code verifier. Must be S256.
ui_locales	YES	BCP47 language tag value. Only “en” and “fr” are supported. This will set the initial language preference for the user within the authorization flow.
login	NO	Indicates that the authorization request should be treated as an existing user and will redirect the user directly to MOA. Its value should be “true” when used.

Client Callback and Authorization Code from the Authorization Server

When the user approves the authorization request, the service redirects the user-agent to the application redirect URI, which was specified during the client registration, along with an Authorization Code.

Based on the redirect URI passed in the authorize step above, here's a sample of a successful response:

https://{client_redirect_url}?code=361e4986-56b2-446f-9718-b3794305f388&state=1654029277

If an error occurred or the end-user did not grant the client access, the Authorization Server will redirect back to redirect_uri provided with an error. Here's a sample of an error response:

https://{client_redirect_url}?error=access_denied&state=1654029277&error_description=Access+denied+by+resource+owner+or+authorization+server

Access Token Request

The client application requests an access token from the API by passing the authorization code that was returned during the callback along with client authentication details. Full details on the client authentication requirements for SoF can be found within Section 4.1.3. Access Token Request and Open ID Connect 1.0 specifications, section 9 - Client Authentication

Request

For illustration purposes, see sample of request:

POST /oauth2/token

```
curl --request POST  'https://{AuthorizationServer_base_url}/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'redirect_uri={Client_redirect_url}' \
--data-urlencode 'code=df4ba24c-ebfb-43b8-90d9-64d90f7dd647' \
--data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \
--data-urlencode 'client_assertion=eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.
_uYEHZzLNEERm_bXr0R6uN-ig-l6bgVNsTvswtvRx7AtlvOb0B62FH1Kp4mLc7W8JoqMkowfV-
UVXKoFFtIOQj6d9omwEHljWPY9BR_YtUVfkGzxTk1S3Ieax0tQT4Ke0ed9kPTviLKAD04D7lrRu2gpPAq4y8
S1UEVshddQmDZEt7WlgmgDkK5pw'
```

PARAMETER	REQUIRED	DESCRIPTION
grant_type	YES	Must be 'authorization_code'
redirect_uri	YES	The redirect URI that was sent with the authorization request in 2.2.1
code	YES	Actual authorization code value that was received in 2.2.2
client_assertion_type	YES	Must be set to urn:ietf:params:oauth:client-assertion-type:jwt-bearer
client_assertion	YES	Signed JWT (see 2.1.1 Client Authentication)
code_verifier	YES	The code challenge from 2.2.1 is the BASE64-URL-encoded string of the SHA256 hash of the code verifier.

Response

The endpoint will return the access token to be used against the resource server to access protected resources along with its expiry, a refresh token and what scopes were granted as part of the authorization request. For illustration purposes, see sample of response below:


```
{
  "access_token": "eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.-
zF1qIz34D1X_U1k2fApunTD-sScc525GwYhjixMQ174a8DPRArjkCn2ihBzB-
KNXk3eSF1J8GkcfePjG35IalbldTKn23u3MeAUJSUPtMdOhXoeShOEp4uu4vBzPHU4VOayRizKsT0TrVwi7
919_QDMRCQ",
  "expires_in": 3599,
  "token_type": "Bearer",
  "refresh_token": "e0dd0169-36a9-4cf4-b30a-3b78c6162bc3",
  "scope": "openid launch/patient patient/patient.read",
  "patient": "e800c98f-8c4a-41a4-bc72-c95f97189f2e"
}
```


Refresh Token Request

The access tokens provided by the token endpoint will expire quickly. In order to receive new access tokens, a subsequent request to this endpoint can be made using the refresh token.

Request

For illustration purposes, see sample of request below:

POST /oauth2/token

```
curl --request POST 'https://{authorization_server_base_url}/oauth2/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'redirect_uri={Client_redirect_url}' \
--data-urlencode 'refresh_token=df4ba24c-ebfb-43b8-90d9-64d90f7dd647' \
--data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \
--data-urlencode 'client_assertion=eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.
UVXKoFFtIOQj6d9omwEHljWPY9BR_YtUVfkGzxTk1S3Ieax0tQT4Ke0ed9kPTviLKAD04D7lrRu2gpPAq4y8
S1UEVshddQmDZEt7WlgmgDkK5pw'
```

PARAMETER	REQUIRED	DESCRIPTION
grant_type	YES	Must be ‘refresh_token
redirect_uri	YES	The redirect URI that was sent with the authorization request in 2.2.1
refresh_token	YES	The refresh token received from a previous call to the token endpoint
client_assertion	YES	Signed JWT (see 2.1.1 Client Authentication)
client_assertion_type	YES	Must be set to urn:ietf:params:oauth:client-assertion-type:jwt-bearer

Response

RESPONISE PARAMETER	DESCRIPTION
access_token	The OAuth 2.0 token used against APIs to retrieve protected resources
refresh_token	The refresh token to be used to get an updated access token after the initial one has expired
token_type	bearer
expires_in	Seconds before the token will expire
scope	The resource server specific scopes that were granted as part of the authorization request for the user

Requesting Health Records

The method for a client to know where and how the patient’s Health Record is accessed is out of scope. The access tokens grant access to the Health Record through the Health Gateway Application. The patient identifier returned through the launch/patient MAY be needed as a query parameter or as part of a URL to specify the correct patient whose data is being requested. The Health Gateway may automatically scope requests to the patient in context, or may reject requests, which do not explicitly include the patient identifier.

Token Revocation

Overview

Token Revocation enables clients to revoke an access token they no longer need. Token Revocation disables the submitted token, refresh token, and any other associated access tokens that are issued. Once a token gets revoked, it will no longer serve to access the Resource Server API, as the token will fail introspections at the Authorization Server.

A client can revoke a token by sending an access token or the associated refresh token to the defined endpoint, that is POST /oauth2/token/revoke

The endpoint requires client authentication using a JWT client assertion as described above.

Request

For illustration purposes, see sample of request:

POST /oauth2/token/revoke

```
curl --request POST 'https://{authorization_server_base_url}/oauth2/token/revoke' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'token=Ahj1jdsajkhcuh122vhbjdsabhv' \
--data-urlencode 'token_type_hint=access_token' \
--data-urlencode 'client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \
--data-urlencode 'client_assertion=eyJraWQiOiJyc2ExIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.
_uYEHZzLNEERm_bXr0e0ed9kPTviLKAD04D7lrRu2gpPAq4y8S1UEVshddQmDZEt7WlgmgDkK5pw'
```

PARAMETER	REQUIRED	DESCRIPTION
token	YES	An access token or refresh token that is retrieved from the token endpoint
token_type_hint	YES	Must be one of: ‘access_token’, ‘refresh_token’
client_assertion	YES	Signed JWT (see 2.1.1 Client Authentication)
client_assertion_type	YES	Must be set to urn:ietf:params:oauth:client-assertion-type:jwt-bearer

Response

A successful revocation will return a simple HTTP 200 response code with an empty response body.

Access / Allow-listing

Overview

Access to non-production environments is restricted by IP allow-listing. To access these environments, the list of IP addresses will need to be provided to gain access. This should include end users who will be testing the solution as well as any systems that make any of the HTTP requests outlined in this document.

In some situations, an integrator may need to configure their local network appliances to direct traffic to egress through a known endpoint to facilitate this allow listing. Please consult your network provider to identify if this is required. In these cases, access to the following domains will be added to this list:

PST (SANDBOX) ENVIRONMENT

https://pst.myaccount.ontariohealth.ca

https://consent.pst.myaccount.ontariohealth.ca

https://idstore.pst.myaccount.ontariohealth.ca

https://authserver.pst.myaccount.ontariohealth.ca

https://fhir-rsa.pst.myaccount.ontariohealth.ca

PPE (TRAIN) ENVIRONMENT

https://ppe.myaccount.ontariohealth.ca

https://consent.ppe.myaccount.ontariohealth.ca

https://idstore.ppe.myaccount.ontariohealth.ca

https://authserver.ppe.myaccount.ontariohealth.ca

https://fhir-rsa.ppe.myaccount.ontariohealth.ca

Appendix

My Ontario Account for Health Account Behaviors

While not directly relevant to the integration described in this document, it can be helpful to be aware of how user sessions are managed within the My Ontario Account for Health (MOAH), and what a user will experience when an authorization request is initiated including when they may be prompted to take action to consent to share data or re-authenticate themselves.

When an authorization request is made against the system a check will first be done to see if the user currently has an active session with the system. If an active session is present, a second check will be done to see if the user has authorized the same client within the last year. If so, the user will be immediately redirected back to the relying party that initiated the authorization request.

If the user has a session but it has been more than a year since the last authorization, or authorization has not yet been given the user will be prompted to agree to share the data.

If the user does not yet have a session, they will be prompted to either login or create a new account. See below for the behaviors related to session management within the Health Account.

Session Behaviors: Definitions

MOAH Session

The MOAH has one session in each single browser. A session is established after a login event and is bound to a browser. The session is established based on:

- an identifier stored on the browser and server
- MOAH Session Expiration Time
- MOAH Max Session Extension Time

Note: A user will have different session sessions in different browsers but many tabs within one browser will share the same session. Terminating the browser session will cause the browser to lose association with the session stored at the server.

Note: The user session has no effect on access and refresh tokens as described in the integration portion of this document. These tokens may last longer or shorter than a user's session.

MOAH Session Expiration Time

When the current browser session will end. If this time passes with no browser-based interaction with the MOAH, the session expires, and a user must login to establish a new session. This is the effectively the 'SSO window', a user can open a new app within this time without login.

MOAH Session Extension

The MOAH session will be extended whenever the user's browser (front-channel) interacts with the MOAH. When the user's browser returns to MOAH, we determine that the user is still active in that browser. Interaction outside of the MOAH will not affect the session. Any action that a MOAH user with an active session takes in their browser. This includes:

- Navigating between pages
- Clicking on links, buttons or other interactable elements within a page
- Scrolling up or down on the page
- Authorization within one year of previously given authorization

MOAH Max Session Extension Time

On the first login we establish a maximum time that a session can be extended until. If an extended session would exceed this time, the session time is set to the max session extension time.

Glossary

TERM	DESCRIPTION
JWT (JSON Web Token)	JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object
Access Token	An access token is an opaque string issued by an authorization server that grants access to resources.
Client, Relying Party, Service Provider	A component that in an OAuth context requests authorization to access restricted resources.
Scope	Is an identifier that mutates the permissions granted as part of an OAuth authorization request. As an example, a scope could be “email” which may imply access to an email address, while the scope “write” may imply write access to a resource.
Resource Server	A component that holds restricted resources that requires authorization before being accessed.
Resource	A specific resource that a resource server makes available for access with authorization. A resource is of a specific instance of a resource definition that is made available.
Resource Definition	A specific type of resource being offered, as an example “identity” or “health data”

Reference Material

TITLE	URL
OAuth2 Standard	https://tools.ietf.org/html/rfc6749
OIDC Standard	https://openid.net/specs/openid-connect-core-1_0.html
SMART On FHIR Standard	http://www.hl7.org/fhir/smart-app-launch/
DHI Description of Activities	Digital Health Identifier: Description of Activities | Ontario Health
Digital Health Identifier Consent Policy and Procedure	Digital Health Identifier Consent Policy and Procedure
Digital Health Identifier Policy and Procedure for Collecting Personal Health Information	Digital Health Identifier Policy and Procedure for Collecting Personal Health Information | Ontario Health
Digital Health Identifier Request for Access to Personal Health Information Policy and Procedure	Digital Health Identifier Request for Access to Personal Health Information Policy and Procedure | Ontario Health
Digital Health Identifier Retention Standard	Digital Health Identifier Retention Standard | Ontario Health

Sample OIDC Metadata

GET /.well-known/openid-configuration


```
{
   "authorization_endpoint": " https://{authorization_server_base_url}/transaction/authorize ",
   "token_endpoint": " https://{authorization_server_base_url}/oauth2/token ",
   "registration_endpoint": " https://{authorization_server_base_url}/oauth2/register ",
   "introspection_endpoint": " https://{authorization_server_base_url}/transaction/introspect ",
   "issuer": " https://{authorization_server_base_url}",
   "jwks_uri": " https://{authorization_server_base_url}",
   "response_types_supported": [
      "code",
      "id_token"
   ],
   "grant_types_supported": [
      "authorization_code",
      "refresh_token",
      "client_credentials"
   ],
   "code_challenge_methods_supported": [
      "S256"
   ],
   "token_endpoint_auth_methods_supported": [
      "client_secret_basic",
      "client_secret_post",
      "private_key_jwt"
   ],
   "token_endpoint_auth_signing_alg_values_supported": [
      "PS384",
      "ES256K",
      "ES384",
      "RS384",
      "ES256",
      "RS256",
      "ES512",
      "PS256",
      "PS512",
      "RS512"
   ],
   "subject_types_supported": [
      "public"
   ],
   "userinfo_endpoint": " https://{resource_server_url}/api/userinfo",
   "request_uri_parameter_supported": false
}
```

Need this information in an accessible format? 1-877-280-8538, TTY 1-800-855-0511, info@ontariohealth.ca.
Document disponible en français en contactant info@ontariohealth.ca