Payer-to-Payer Connectivity Hub

Companion Guide

A guide for payers implementing Availity's Payer-to-Payer APIs for regulatory compliance with CMS-0057.

Disclosure Statement

Availity provides the information in this document for education and awareness use only. While Availity believes all information in this document to be correct at the time of writing, this document is intended for educational purposes only and does not purport to provide legal advice. If you require legal advice, you should consult with an attorney. The information provided here is for reference use only and does not constitute the rendering of legal, financial, or other professional advice or recommendations by Availity. The listing of an organization in this document does not imply any sort of endorsement, and Availity takes no responsibility for the third-party products, tools, and Internet sites listed. The existence of a link or organizational reference in any of the following materials should not be assumed as endorsement by Availity.

Contents

Introduction

The Problem: Payer-to-Payer Data Exchange

Over time, most people experience their health insurance moving from one payer to another due to a variety of reasons: changing jobs, retiring, selecting a new Medicare Supplemental plan, getting married and consolidating benefits to a spouse's employer, among other reasons. To have the most complete patient record possible, the new healthcare plan must be able to import patient data from the previous plan.

CMS-regulated payers are mandated by CMS-0057-F to exchange certain patient clinical data (specifically the US Core Data for Interoperability (USCDI) data set, version 1) at the patient's request, enabling the patient to take their information with them as they move from one payer to another. This ensures that the current payer has a cumulative health record for the individual, facilitating informed decision-making, efficient care, improved risk stratification for the payer, and better health outcomes. Payer-to-Payer Data Exchange is also simply referred to as **PDex**.

This document follows the terminology used by the HL7 PDex Implementation Guide:

New Payer refers to a patient's new or current healthcare plan, the entity that will provide coverage for the patient going forward.

Old Payer refers to the patient's prior or soon-to-be-prior healthcare plan. There may be some period of overlap or dual coverage, but in general it is expected that the New Payer will become the plan of record for the member.

Member or **Patient** refers to the individual whose data is being requested from the Old Payer by the New Payer.

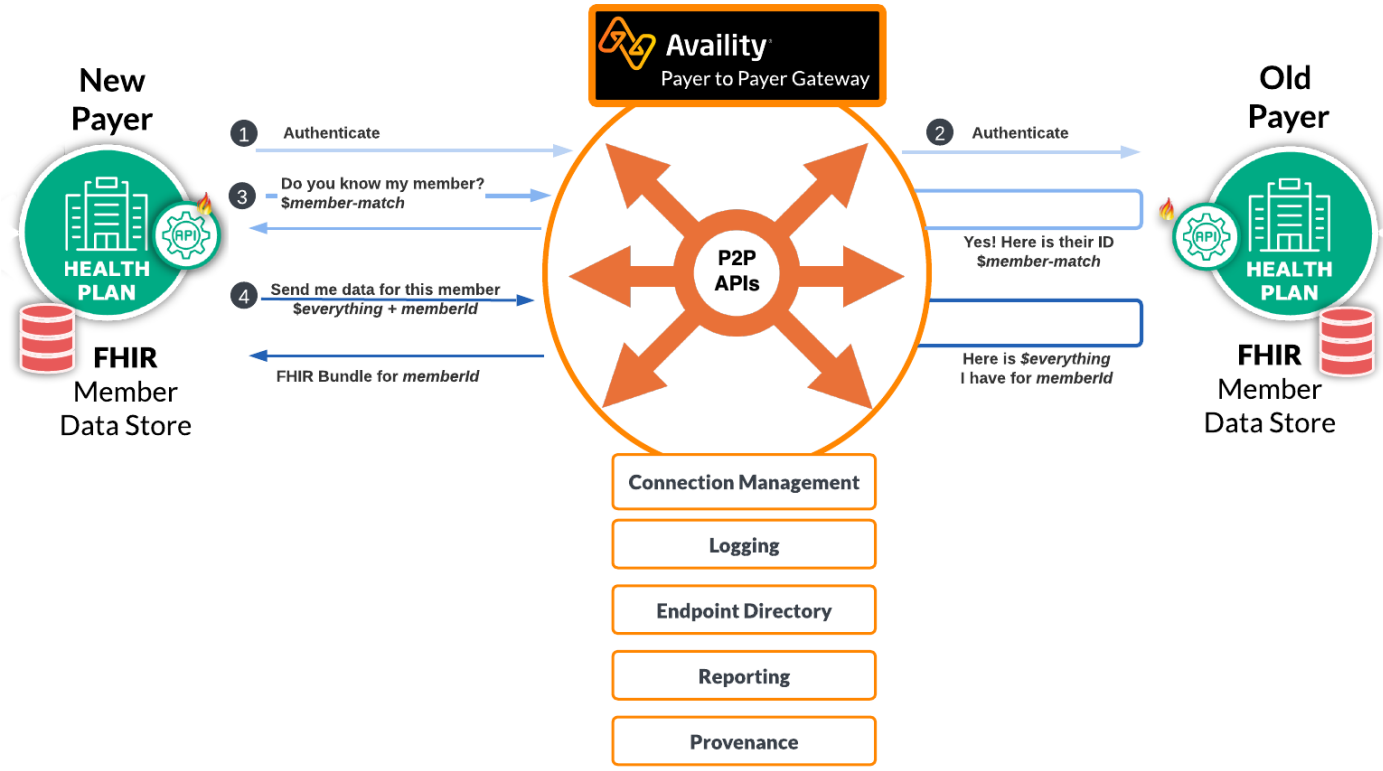
How does Availity solve for PDex?

For a given Payer to establish relationships with, then construct, test, and deploy interfaces to and from every other Payer with whom they might need to exchange data (a “many to many” architecture), as well as maintain all that code going forward would be a daunting task - slow, difficult, costly, and horribly inefficient. Fortunately, Availity has a solution.

The Availity Payer-to-Payer Connectivity Hub

Leveraging our existing relationships with the nation’s top Payers and our technical expertise, Availity is uniquely situated to facilitate the exchange of patient data between Old Payer and New Payer, serving as an intermediary via a “hub and spoke” architecture. Rather than every New Payer needing to communicate with every Old Payer (and vice versa), New Payers can simply request a patient’s data from Availity’s service with a handful of basic FHIR API calls. Availity will communicate with the Old Payer, retrieve the data indicated, and provide it to the requester.

NOTE	This documentation assumes that the patient requesting their New Payer obtain data from their Old Payer knows who their previous healthcare plan was and has provided their member ID and the other information required to find their data in the Old Payer’s system. If the patient does not know their previous healthcare plan membership, that is considered a separate use case that will be supported in the future.



Overview

Exchange Modalities

Single Patient

Exchange data for one member at a time. Results are returned in the response body as JSON-formatted data.

Bulk (Multiple Patients)

In this mode, data for multiple members may be requested in a single operation by submitting a list of patient IDs. Availity will handle proxying the requests to retrieve data for the listed members. The response to a bulk request includes three lists per the HL7 FHIR Davinci implementation guide:

1. Matched Member Group – a list of patients matched by the Old Payer with links to download their data as NDJSON file(s)
2. Not Matched Group – a list of patients who could not be identified by the Old Payer
3. Consent Constrained Group – a list of patients matched by the Old Payer with links to download their data as NDJSON file(s), but whose returned data is limited by the patient's Consent for data sharing.

NOTE: This is expected to be the most common mode of operation, since it can encompass the Single Patient case by including only one member in the bulk request's patient list.

Querying a Patient's Resources Individually

Coming soon. Rather than requesting all resources for a patient, this mode permits you to request only a specified type of resources for the member. When there is a specific type of information that needs to be exchanged between payers, this operation would save transferring unnecessary or undesired resources. This mode would be used if, for example, you wanted to retrieve *only* an individual's Encounters, or *only* Allergy information, etc.

Concurrent Coverage and the **_since** Parameter

While Concurrent Coverage does not represent a use case in the manner that the three scenarios above do, it does represent an important application of the Payer-to-Payer Data Exchange APIs.

Sometimes, a patient may have two (or more) active health insurance plans simultaneously. For example, a senior might have both Medicare and a commercial supplemental insurance plan. This situation is known as **"concurrent coverage."** Concurrent payers face a unique challenge in that they must receive ongoing updates about the member they share with another payer.

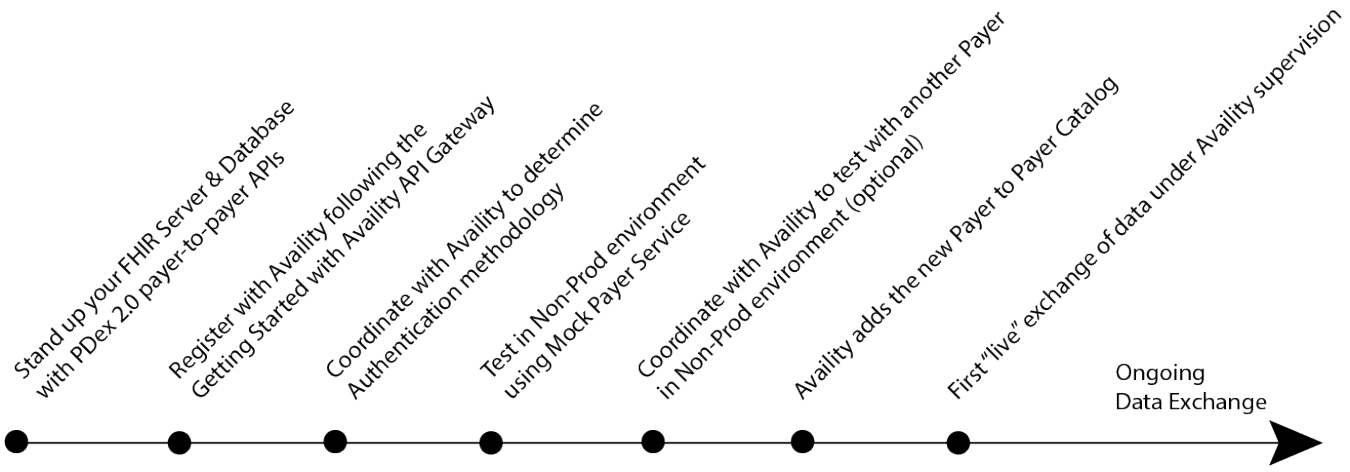
Concurrent payers interact with **Connectivity Hub** using the same sequence of API calls as a typical one-time data exchange. However, they will want to use the **"_since"** parameter in their subsequent requests. This allows them to specify the date of their last exchange, ensuring they receive only **incremental updates** rather than a full export of the patient's data.

For example, if a member had a mammogram last month paid by Medicare, the other plan is notified and does not incorrectly flag it as an open gap in care. This prevents unnecessary reminders urging the member to get a mammogram. When checking for updates, concurrent payers primarily seek **new patient data** since their last query. Using the **"_since"** parameter, they can request updates in a format like: *"Only provide data since January 1, 2025."*

Implementation Plan

The steps required for implementation depend on the state of the payer in question, but the following sequence must occur onboard and implement Availity's Payer-to-Payer data exchange solution.

1. Onboarding Objectives:
    - Confirm whether the customer is onboarding solely to the Payer-to-Payer Connectivity Hub or also to the Onyx solution.
    - Determine if the payer is contracting as a new customer or engaging through an existing customer relationship.
    - Identify the onboarding methodology: bulk submission or single transactions.
    - Clarify whether the payer is onboarding as a new entity, a returning entity, or both.
    - Validate payer prerequisites and share the companion guide along with relevant onboarding resources.
    - Establish the authentication method (e.g., OAuth2 using client ID and secret).
2. Payer implements a FHIR 4.0+ server/database with PDex v2.0 or higher payer-to-payer API(s).
3. Payer follows the "Getting started with Availity API Gateway" steps to register with Availity (non-production environment, production environment, or both)
4. Authentication methodology determined
5. Payer tests in the non-production environment with mock payer service (optional but recommended)
6. (optional) Payer works with Availity to test with another cooperating payer in non-production environment.
7. Availity's Payer Catalog is updated with the onboarding payer's information.
8. Payer makes first "live" exchange of data under Availity supervision.



Payer-to-Payer Connectivity Hub Features

The table below highlights the key features of Availity's Payer-to-Payer Connectivity Hub.

Feature	Description
Authentication, Trust, and Access to Payer Network	A secure connection with a valid and trusted healthcare payer to enable exchange with a grid of other connected payers.
Endpoint Catalog	A discrete, up-to-date list of networked payers including parent company, plan details, and unique ID is made available via API call for real-time app integration.
Dynamic Request Routing	Through a single connection, a payer may send requests to any other payer in the network using the target payer's identifier in the CoverageToMatch element of the request. The target payer's identifier can be found in the Endpoint Catalog.
Synchronous Data Retrieval **(Member Match, Token Request, $everything)**	Availity serves as an intermediary, brokering transactions between parties and preventing each party from seeing internal details of the other party. Requests are submitted to Availity's service, not to the end party. Likewise, responses are sent to Availity first, not directly to the original requestor. For an individual member request, Availity maps requests, request parameters, responses, tokens, and URLs so that the end customer can respond without concern of exposing sensitive information. Availity will continue working with its network to manage data governance issues regarding these transactions.
Asynchronous Data Retrieval **(Member Match, Token Request, $everything)**	Bulk requests (i.e., a group of members rather than one) are handled by submitting a list of patients. As in Synchronous Data Retrieval, Availity serves as an intermediary broker. Availity maps requests, request parameters, responses, tokens, and URLs so that the end customer can respond without exposing anything to the originating requestor. Availity works continually with its network to manage data governance issues regarding these transactions.
Mock Payer Service	Network participants can test the API using calls to a synthetic payer. This mock payer supports various transactions with synthetic members thereby avoiding any PHI/PII concerns. New members can exercise all the Payer-to-Payer's API functionality before engaging with actual Payers, permitting full end-to-end testing before full implementation.

Feature	Description
Transactional Reporting	Reports demonstrating network size, transactional volumes, and validation issue trends will be available to participants and will be used to address deficiencies observed in participants.
Operational Monitoring	Availity will track and verify service availability and uptime with its network.
Payload Validation	Although validation errors within a payload will not typically prevent them from being delivered, Availity will capture validation issues to identify trends and bad actors. This information will be used to proactively address these concerns and improve network performance.
Provenance	Availity will supplement payloads with FHIR Provenance resources that record the source payer and Availity as an intermediary.
Documentation and Guides	In addition to adhering to mandated and recommended standards, Availity provides world-class guidance on interacting with our service. This may include guides, access to a documentation portal, training videos, 1:1 consultation, and workshops.
Payer Onboarding and Support	Availity will be the front door and initial service department for any payers connecting to the network, whether as a new payer, old payer, or both in the Payer-to-Payer use case.

Patient Matching

Matching criteria is determined by each payer individually but generally will match on a known member ID if available as a primary indicator and use member demographics for reinforcement. At a minimum, the values below should be provided for any member-match request.

- Old Plan Member ID
- Last Name
- First Name
- Date of Birth
- Phone number
- Address (complete address or just postal code)

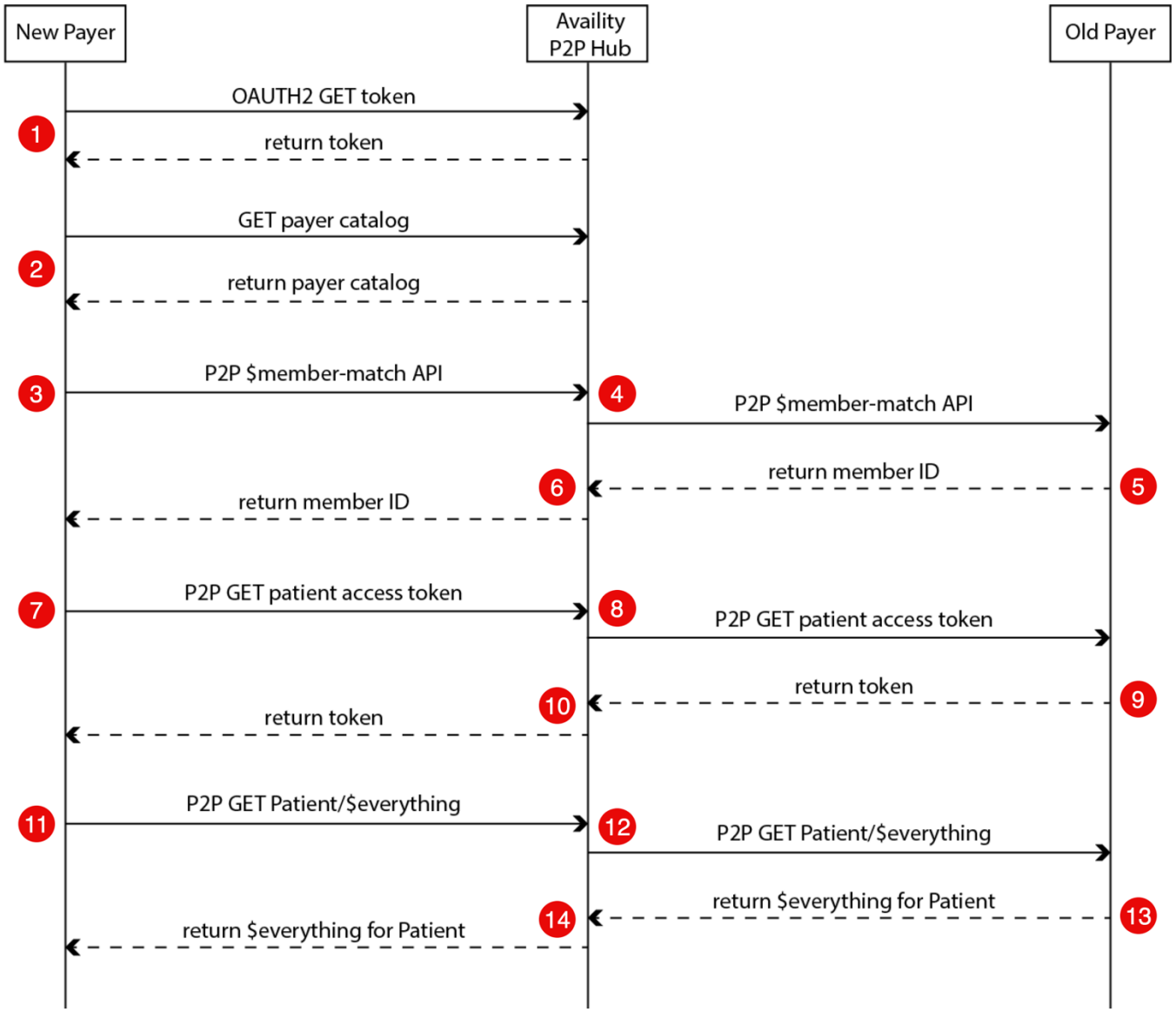
Technical Overview – Single Transaction

Availity establishes connectivity with payers in advance and manages authentication exchanges with old payers. Payer-to-Payer Connectivity Hub acts as a proxy, preventing direct access between the two parties. It also obscures internal details, such as direct URLs to the old payer's FHIR server. This section describes the exchange of requests, responses, and maps this to a network sequence diagram.

1. New Payer authenticates with Availity's Payer-to-Payer Connectivity Hub using onboarding credentials to obtain an access token.
2. New Payer requests the latest Payer Catalog to identify the correct Old Payer and plan. Availity returns the catalog.

3. New Payer sends $member-match request to Availity, specifying the patient's Old Payer.
4. Availity authenticates with the target Old Payer and routes the request.
5. Old Payer performs member matching and returns the FHIR resource ID for the matched member.
6. Availity proxies the member ID and URLs and routes the response back to the New Payer.
7. New Payer requests a scoped token to access the patient's data.
8. Availity proxies the request to the Old Payer.
9. Old Payer returns the patient access token.
10. Availity proxies the token and URLs to the New Payer.
11. New Payer uses the scoped token to send a GET $everything request for the member's data.
12. Availity proxies the request to the Old Payer.
13. Old Payer returns paginated results, adhering to the bounds of the patient's consent.
14. Availity proxies the $everything response to the New Payer.

The following network-oriented diagram illustrates the sequence of API calls and responses exchanged between the New Payer, Availity's Payer-to-Payer Connectivity Hub, and the Old Payer.

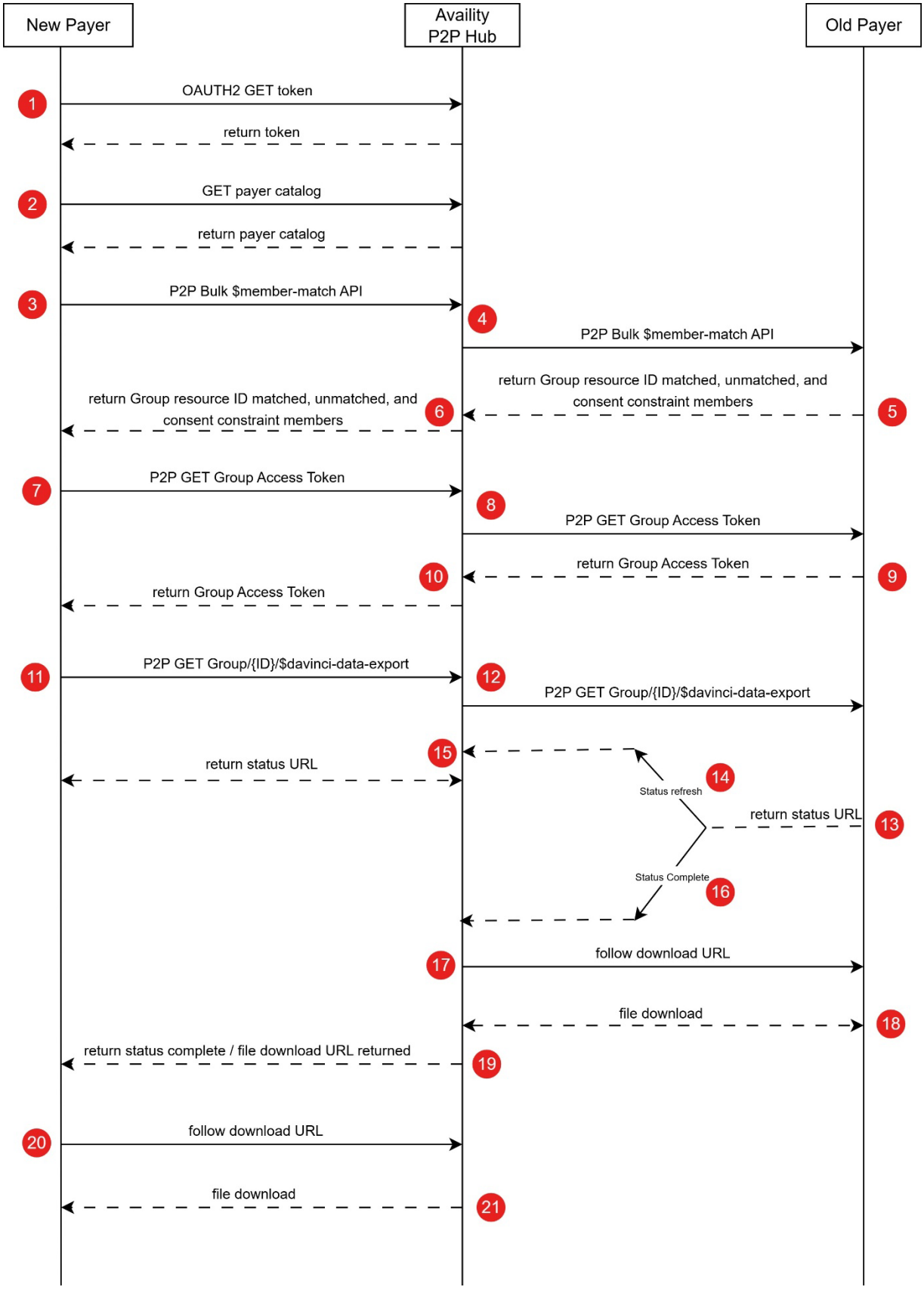


Technical Overview – $multi-member-match request

Availity establishes connectivity with payers in advance and manages authentication exchanges with Old Payers. The Payer-to-Payer Connectivity Hub acts as a proxy, preventing direct access between the two parties and obscuring internal details such as direct URLs to the Old Payer's FHIR server.

This section explains the exchange of requests and responses and illustrates the process in a network sequence diagram.

1. The New Payer uses the credentials provided during onboarding to authenticate with Availity's Payer-to-Payer Connectivity Hub and retrieve an access token.
2. The New Payer requests the most recent Payer Catalog to identify the correct Old Payer and plan. Availity returns the catalog.
3. The New Payer submits a bulk $multi-member-match request to Availity, identifying a list of patients linked to a single Old Payer.
4. Availity authenticates with the designated Old Payer and forwards the request.
5. The Old Payer matches members and returns a FHIR Group resource ID that includes matched, unmatched, and consent-constrained members.
6. Availity proxies the URLs and Group ID and forwards the response to the New Payer.
7. The New Payer requests a scoped token to access data for the matched group.
8. Availity proxies the request to the Old Payer.
9. The Old Payer returns a Group Access Token (also called a Group Scoped Token).
10. Availity proxies the token and URLs and forwards them to the New Payer.
11. The New Payer uses the scoped Group Access Token to submit a GET $davinci-data-export request for all data associated with the group's members.
12. Availity proxies the URLs and routes the request to the Old Payer.
13. The Old Payer returns a status URL that Availity uses to monitor the progress of bulk export file preparation.
14. Availity proxies the URLs and routes the $davinci-data-export response to the New Payer and continues polling the status endpoint at regular intervals.
15. Availity returns a status URL to the New Payer who checks that status endpoint until complete.
16. The Old Payer polls the endpoint, detects the **Complete** status.
17. Availity receives and follows the download URL(s).
18. Availity downloads the file(s).
19. Availity updates the New Payer status to complete with a download URL.
20. New Payer follows the download URL.
21. New Payer downloads the file(s)



Get Started with the Availity API Gateway

The Availity API Gateway provides a unified point of access to Availity's externally facing APIs. This section is intended for developers who are new to the API Gateway and need to complete initial registration, application setup, and subscription tasks.

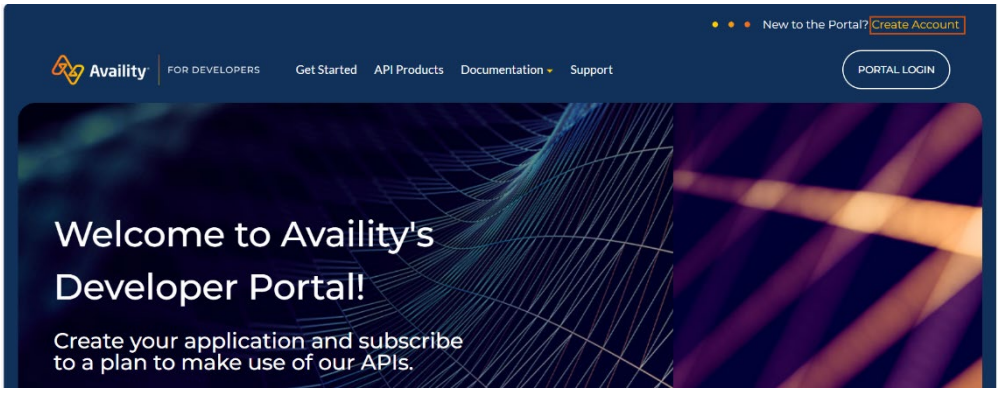
NOTE	Some screenshots in this section are intentionally shortened or truncated to improve readability and avoid unnecessary repetition. Nothing that requires your attention or interaction is omitted. If an element is not shown in a screenshot, you can safely disregard it.

Create an account

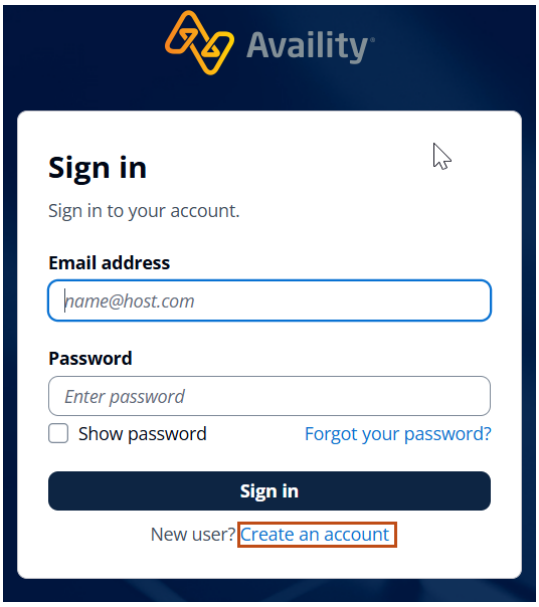
To access the API Gateway, you must register in the Availity Developer Portal.

1. Open a web browser and go to one of the following URLs:
   - **QA environment:** https://qua.developer.availity.com/
   - **Production environment:** https://developer.availity.com/

2. Select **Create Account**.



   - In the **Sign In** dialog box, select **Create an account**.



- Enter the required information:
    - Email address
    - Given name
    - Family name
    - Password
    - Confirm password
- Select **Sign up**.

Set up multi-factor authentication

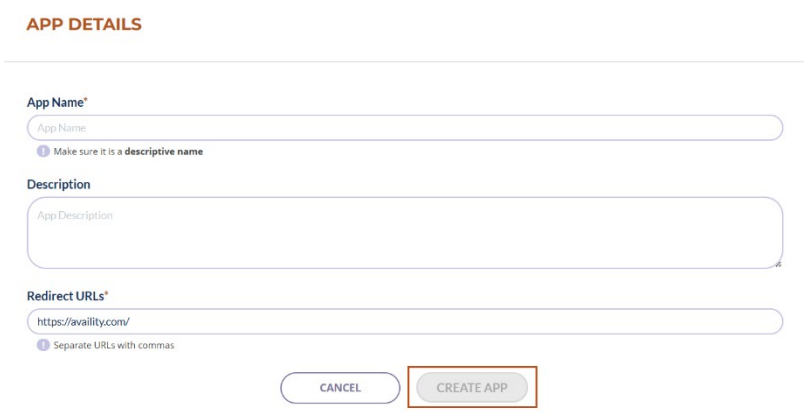
After you create your account, complete multi-factor authentication (MFA).

1. Install an authenticator app on your device.
2. Register your email address in the authenticator app.
3. Scan the QR code displayed on the screen.
4. In the authenticator app, generate a verification code.
5. Enter the verification code on the Multi-factor authentication (MFA) screen.
6. Select Sign in to complete setup and access the portal.

Create an app

After signing up, create an application to associate with your API subscriptions.

1. Sign in to the Availity Developer Portal.
2. From the navigation menu, select **My Apps**.
3. Select +CREATE A NEW APP
4. Enter the following information:
   - App Name
   - Description
   - Redirect URLs
5. Select CREATE APP



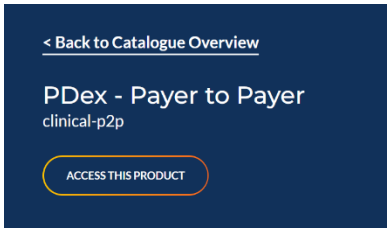
Subscribe to the PDex API

To use the Provider Data Exchange (PDex) APIs, you must request a subscription.

1. Sign in to the Availity Developer Portal.
2. From the navigation menu, select API products.
3. Search for PDex. The following products will display
   - PDex – Payer to Payer
   - PDex – Payer to Payer - Payer Catalog
   - PDex - Payer to Payer Bulk
4. Select **More info** for the desired API product

5. Select **Access This Product** to view the available subscription plans.

   Note: You must submit a subscription request and receive approval before you can use the API product.



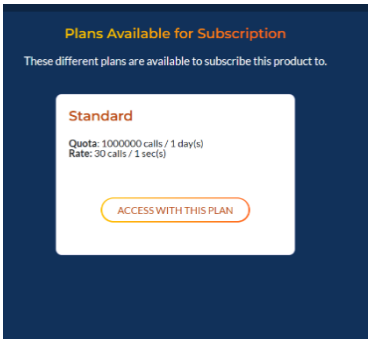
Subscription plan details:

- **Demo Plan**
  Provides a sandbox environment for testing communication with Availity APIs. Demo subscriptions are automatically approved and are recommended for learning the Availity REST API workflow.
- **Standard Plan**
  Provides full access to Availity REST APIs and production data.

Submit a subscription request

After selecting a subscription plan, submit your request.

1. From the subscription notification, select **ACCESS WITH THIS PLAN**.



2. Select **Go to cart** to complete the subscription request.

NOTE	Alternatively, you can select the cart icon next to your username.

3. Verify that the correct API product displays in the **Selected Products** section.
4. Verify that the correct subscription plan displays in the **Selected Plan** section.
5. In the **Select an App** section, select either of the following:
   - **Create a New App** and then enter the application details, or
   - **Existing App** and then select an existing application.
6. Select **SUBMIT REQUEST**.
7. The Trading Partner Management team will assist with contracting and activation.
8. After contract verification, Availity approves the subscription. You can then view access and credential information on the ***My Apps > Approved Access*** tab.

Testing with Availity's Mock Payer Service

To facilitate initial testing of the Payer-to-Payer APIs, Availity offers a Mock Payer Service. The Mock Payer Service is intended to validate that Payers can connect to the Availity Payer-to-Payer Connectivity Hub and make all the necessary API calls in an interoperable manner. If you seek to test additional use cases beyond the initial scenario of retrieving a single patient's data from an old payer, please contact Availity.

Payers should plan to test using the Mock Payer Service initially, and then to conduct system-to-system testing with a cooperative peer in the non-production environment, before moving to a live Production environment. Availity can assist in arranging peer-to-peer testing with an appropriate and willing partner to the benefit of each party.

Scheduling time for testing with Availity in the Non-Production environment is important for all parties. Such cooperative testing ensures our partners have appropriate understanding of the exchange and that all your implementation questions are answered before attempting to go live in the Production environment.

The Mock Payer Service is only available in the non-production environment. Mock Service payer IDs can be discovered through the Payer Catalog; the synthetic payers used for testing all have names that end in "-mock." The Mocker Payer Service does not proxy calls to an old payer. Instead, it returns responses with synthetic data. You can direct API calls to any *-mock payer ID, and the results will be identical.

An API collection is available to help with testing. It accelerates connectivity verification, discovery, and evaluation of platform functionality and performance. New customers can access the API collection through the Availity Clinical Solutions online support hub during onboarding.

Limitations of the Mock Payer Service

The Mock Payer Service is not intended to provide a full end-to-end representation of the entire PDex use case, but to familiarize users with the API calls and prepare them for full implementation by verifying the authorization and token exchange process. Thus, there are several limitations that it is important that users are aware of.

- Regardless of which -mock Payer ID is specified in API calls, the responses come from the mock server and will be identical.
- The sample Patient ID provided may be used regardless of which -mock Payer ID is used.
- The Mock Payer Service does not evaluate any of the demographic or biometric fields in the Patient Match call. Only the provided Patient ID is evaluated.
- The Mock Payer Service verifies that the Consent section is present and will reject a member-match request that does not contain an applicable Consent resource as defined in the referenced implementation guide(s). However, the Mock Payer Service does not evaluate the details or content of the Consent section or adjust its response based on the Consent terms. Although the Mock Payer Service does not consider the Consent section, it **must** be respected by Old Payers in a Production environment.

The Payer Catalog

Availity's Payer-to-Payer Connectivity Hub enables payers to exchange data according to the HL7 PDex standards via FHIR API calls. To access a patient's records, a New Payer needs to identify which Old Payer holds the data and determine the specific plan under which the patient was previously enrolled.

Typically, the individual requesting their data will provide this information, but sometimes they may not have the details of their previous insurer or the plan they were a member of. Regardless, it is necessary for the New Payer to determine the Old Payer's Payer ID and the Plan ID.

In support of this application, Availity makes available its Payer Catalog via RESTful API call. The Payer Catalog provides an on-demand up-to-date listing of all the Payers who participate in Availity's Payer-to-Payer Connectivity Hub, and within each of those Payers, the individual plans supported by each. The Catalog is regularly maintained and updated whenever changes occur in the network of Payers, ensuring our users always have access to the most up-to-date information.

Because the Payer Catalog is available in near-real time at the speed of an API call, and formatted as easy-to-digest JSON, its data can potentially be used to populate a customer-facing mobile app or web page that members can make use of when giving their new insurer permission to retrieve their data

from their old insurer. Offering such a self-service tool makes it easier for new members to transition from their Old Payer to you.

To retrieve the Payer Catalog:

1. Authenticate to obtain access token.
2. Request the Payer Catalog using the access token.

Authenticate to obtain an Access Token

Authenticate with Availity's API gateway to obtain an access token using your **client_id** and **client_secret** values using the following API call:

POST {root_URL}/v1/token

The response includes an **access_token** for use in the GET /payer-catalog call:

```
{
"token_type": "Bearer",
"access_token": "XXXXXXXX-EXAMPLE-XXXXXXXXX",
"scope": "hipaa",
"expires_in": 300,
"consented_on": 1728330235
}
```

Request the Payer Catalog

Place the access_token value into the **Bearer token** field of your GET /payer-catalog call, and send that request to the API gateway:

GET {root_URL}/fhir/r4/PayerCatalog

The response will look like the example below - a block of JSON formatted data containing an array of payers, and if a payer offers multiple plans, a sub-array of those plans. In the example below, we see that there are three sample Payers represented: Centene, Blue Cross Blue Shield, and Humana. Observe that Centene does not offer multiple Plans, so its **plans[]** array is empty. Meanwhile, both Blue Cross Blue Shield and Humana each offer three different Plans, so their plans[] arrays are populated.

```
{
    "payers": [
        {
            "payerId": "centene-mock",
            "payerName": "Centene",
```

```
            "payerDescription": "Test Payer Centene",
            "plans": []
        },
        {
            "payerId": "bcbsfl",
            "payerName": "Blue Cross Blue Shield",
            "payerDescription": "Test Payer BCBS",
            "plans": [
                {
                    "planId": "peachstate12345",
                    "planName": "Blue Cross Blue Shield GA",
                    "planDescription": "Plan A Desc"
                },
                {
                    "planId": "testPlanIDB",
                    "planName": "BCBS Test B",
                    "planDescription": "Plan B Desc"
                },
                {
                    "planId": "testPlanIDC",
                    "planName": "BCBS Test C",
                    "planDescription": "Plan C Desc"
                }
            ]
        },
        {
            "payerId": "humana-mock",
            "payerName": "Humana",
            "payerDescription": "Test Payer Humana",
            "plans": [
                {
                    "planId": "humanaPlanIDA",
                    "planName": "Humana Test A",
                    "planDescription": "Humana Plan A Desc"
                },
                {
                    "planId": "humanaPlanIDB",
                    "planName": "Humana Test B",
                    "planDescription": "Humana Plan B Desc"
                },
                {
                    "planId": "humanaPlanIDC",
                    "planName": "Humana Test C",
                    "planDescription": "Humana Plan C Desc"
                }
            ]
        }
    ]
}
```

Key Information

From these Payer Catalog results, it's easy to locate the Payer (and if needed the specific Plan), and the associated **payerID** and **planId** values for use in subsequent API calls to the Payer-to-Payer Connectivity Hub to retrieve a patient's data from their Old Payer.

Application

Payers may consider using the on-demand Payer Catalog to embed a tool in their member-facing workflow for a member to identify their Old Payer, such as a drop-down list pre-populated with the Payers participating in the Payer-to-Payer Connectivity Hub. The member's selection can then be used to determine the Payer ID the New Payer needs to give Availity to route the request for member data.

Payer-to-Payer Singles API Call Sequence

Overview

Retrieving a single patient's data from the Old Payer requires five API calls:

1. POST /token to obtain an access token.
2. GET /PayerCatalog to identify the Old Payer ID for routing the patient data request to the correct party.
3. POST /Patient/$member-match to obtain a FHIR Patient resource ID for the individual on the Old Payer's system.
4. GET /Patient/{id}/token to obtain an access token granting permission to the specific Patient resource on the Old Payer's system.
5. GET /Patient/{id}/$everything (including the patient access token as a special header) to return the patient's data.

Each step is explained in detail below with examples shown in the Insomnia API tool. If you use Postman instead of Insomnia, expect minor differences in screen layout, environment configuration, and variable naming conventions. However, the overall process and sequence of API calls remain the same.

NOTE 1	The examples provided here assume that you are using the Insomnia API collection that Availity provides. This collection includes pre-configured API calls for various mock patients, pre-populated with the URL for our API gateway (https://qua.api.availity.com/). Nonetheless, please pay attention to the URL path in the screenshots, as the first API call includes "v1/" after the URL, and the remaining three calls include "fhir/r4/" after the URL. We will define a variable for **baseUrl** to make testing a little easier.

NOTE 2	For security, Availity's API gateway limits the duration of authentication tokens to 300 seconds (5 minutes). You will need to complete the entire sequence of calls within this timeframe before your access token expires.

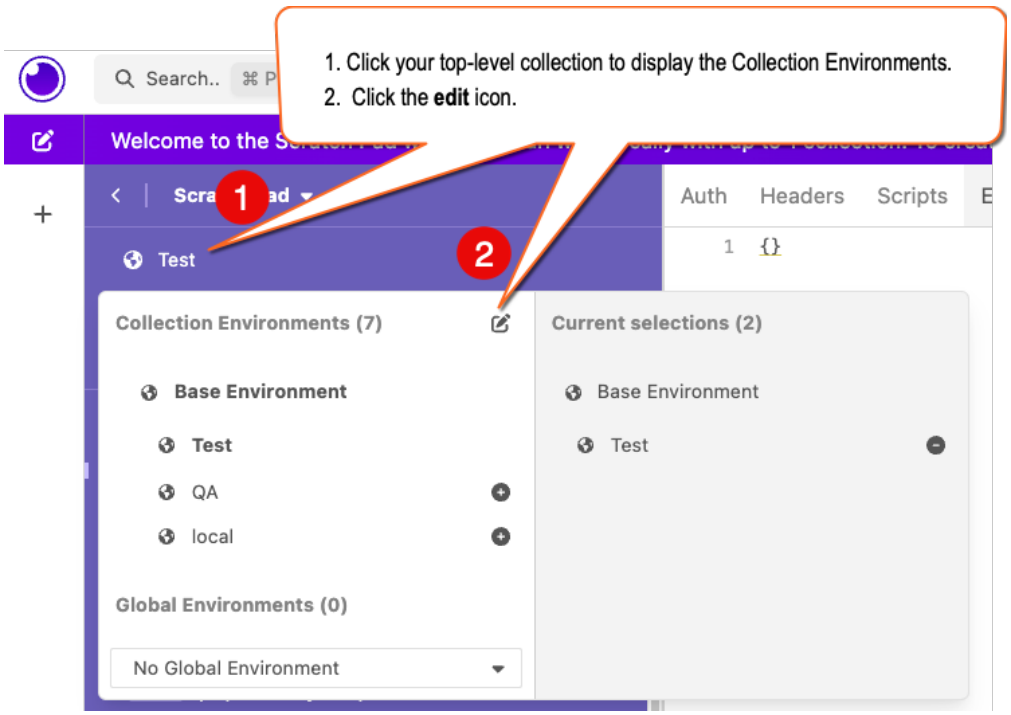
Setting up Variables

NOTE	Using variables is optional but highly recommended to avoid unnecessary manual copy/pasting. This section provides a brief overview of setting up a variable we will use in subsequent API calls. Insomnia's documentation on variables can be found here: https://docs.insomnia.rest/insomnia/environment-variables/

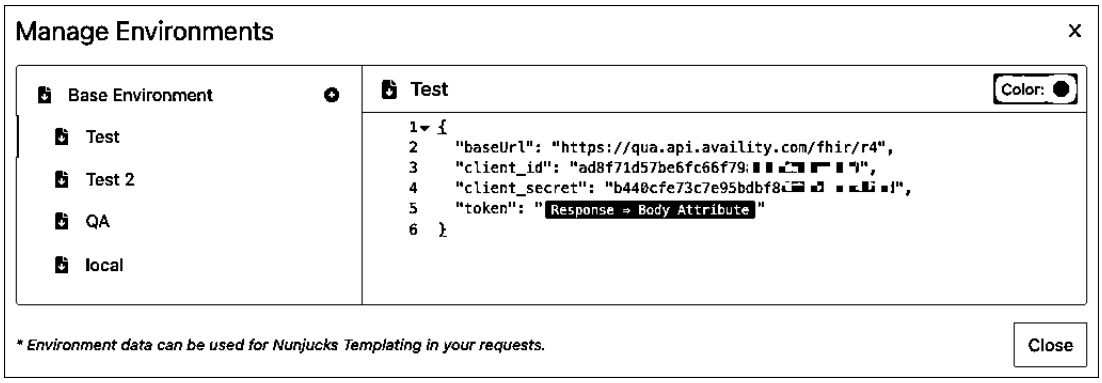
API testing tools such as Insomnia and Postman allow you to define variables that you can populate with fixed values or with a value taken from the response to a previous API call. By using variables with easily remembered names, you can avoid having to manually populate key: value pairs in your API calls, making the testing process easier, faster, and less prone to errors.

When you sign up for Availity's Payer-to-Payer Connectivity Hub via our API Gateway, you will receive a **client ID** and **client secret**. These parameters are unique to each of our customers and must not be shared. You will use these authentication parameters to request an access token from the Connectivity Hub. Your access token is included in all the subsequent API calls you place to the Connectivity Hub. Because these values will not change, we recommend defining variables to represent these fields.

In Insomnia, you define variables by clicking the Settings icon at the top of the API calls list:



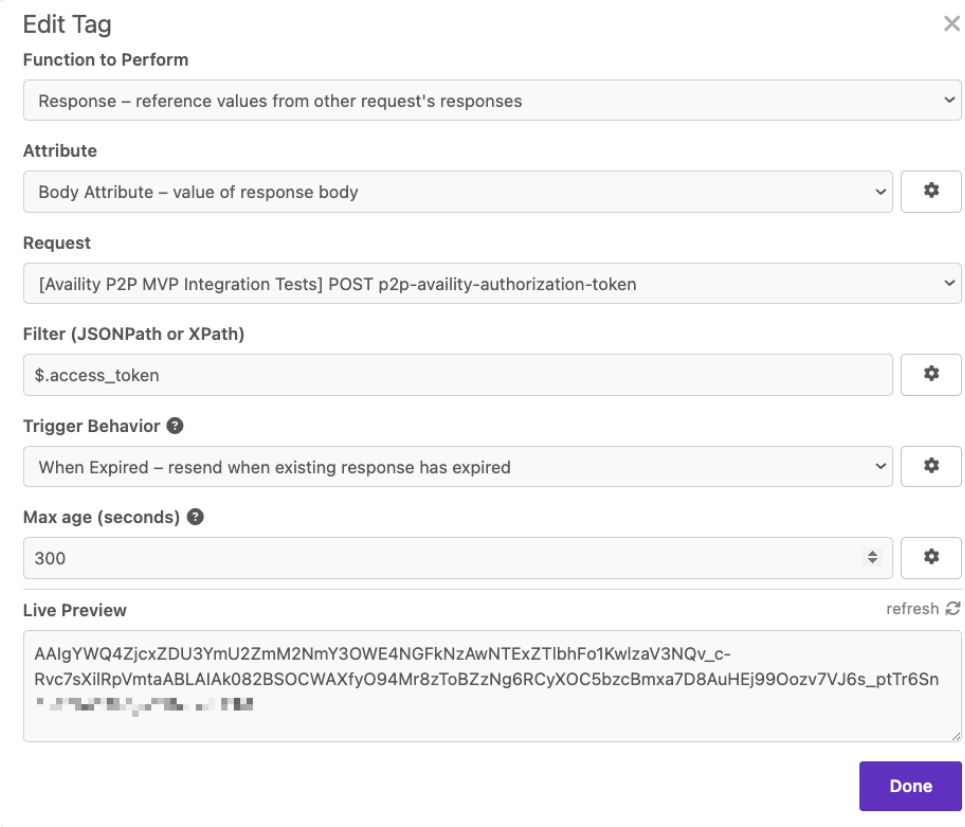
Insomnia's **Manage Environments** screen appears, where variables are defined:



Variables can have a static value, such as baseUrl, or a dynamic definition taking its value from a parameter in the response to a different API call.

Add a Sub Environment if needed ("Test" in the example above), and add lines for **baseUrl**, **client_id**, **client_secret**, and **token**. Populate baseUrl as shown above – https://qua.api.availity.com/fhir/r4. Populate client_id and client_secret with your id and secret. **Token** will take its value dynamically, as we will describe next.

Between the "marks for token, begin typing *Response*. Insomnia pops up a list of options to choose from. Select **Response - Body Attribute**, as above. The tag appears red (undefined); click the red text to open the **Edit Tag** window.



Populate the fields as shown above, and this variable will take its value from the "access_token" field in the response to the **p2p-availity-authorization-token** call, the first call in the PDex sequence. By using this **token** variable in subsequent calls, you will not have to copy/paste the token value repeatedly.

Click **Done**, and the token definition should now appear in blue, indicating that you have defined it correctly. Click **Close** in the Manage Environments screen

1. POST /token

This step authenticates the Payer-to-Payer Connectivity Hub and retrieves the authorization token required for subsequent calls. Before you send this request:

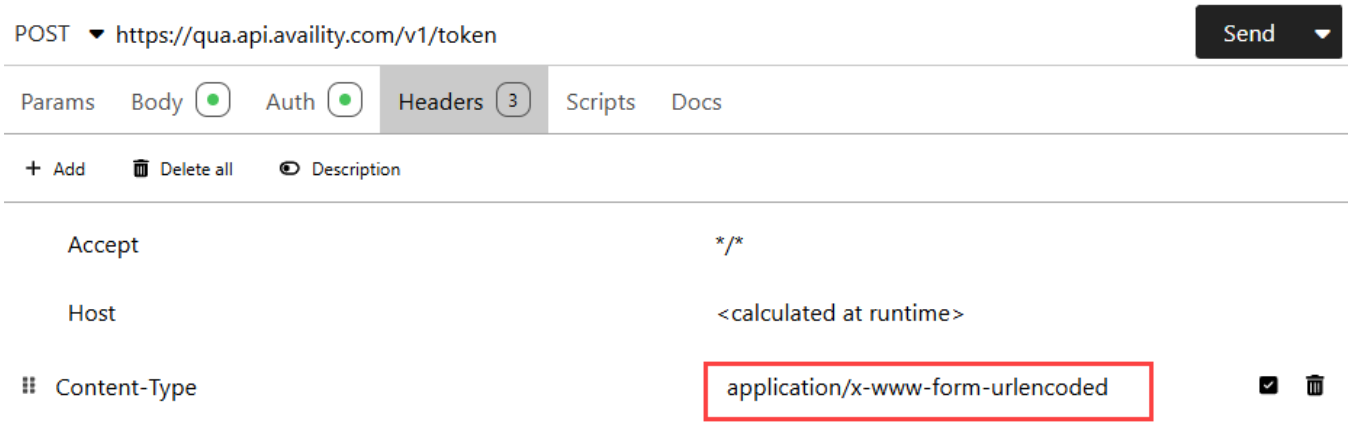
NOTE	The URL for this request is https://qua.api.availity.com/v1. All subsequent requests use the URL defined in the **baseUrl** variable.

1. Set the body type to **Form**.
2. Populate the form data with values for **client_id** and **client_secret**.

Set the Body Type to **Form**

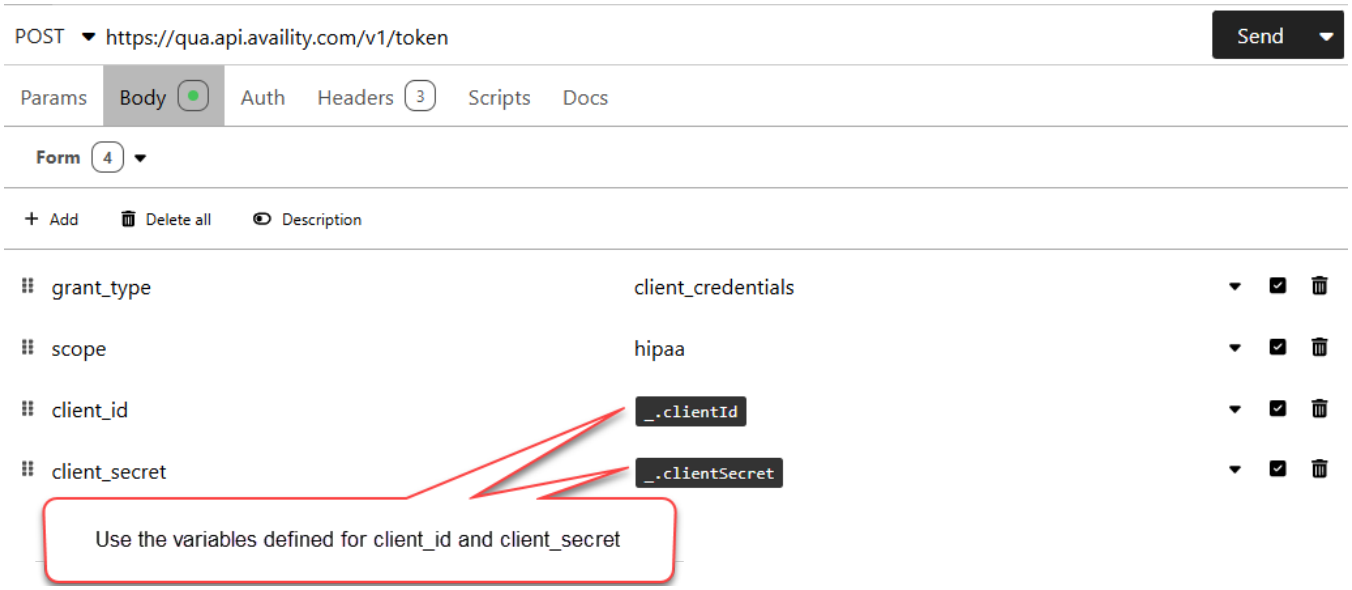
NOTE	This is already done for you in the P2P API collection Availity make available, but it is worth verifying if only to familiarize yourself with the different tabs and components of an API call as represented in Insomnia. Postman's UI is similar.

The POST /token request uses a Form data body, while all other API requests use JSON payloads in the body. In an API tool such as Postman or Insomnia, when you change the body type from JSON to Form, the Headers tab automatically includes a **Content-Type** header with the value **application/x-www-form-urlencoded**, as shown below:



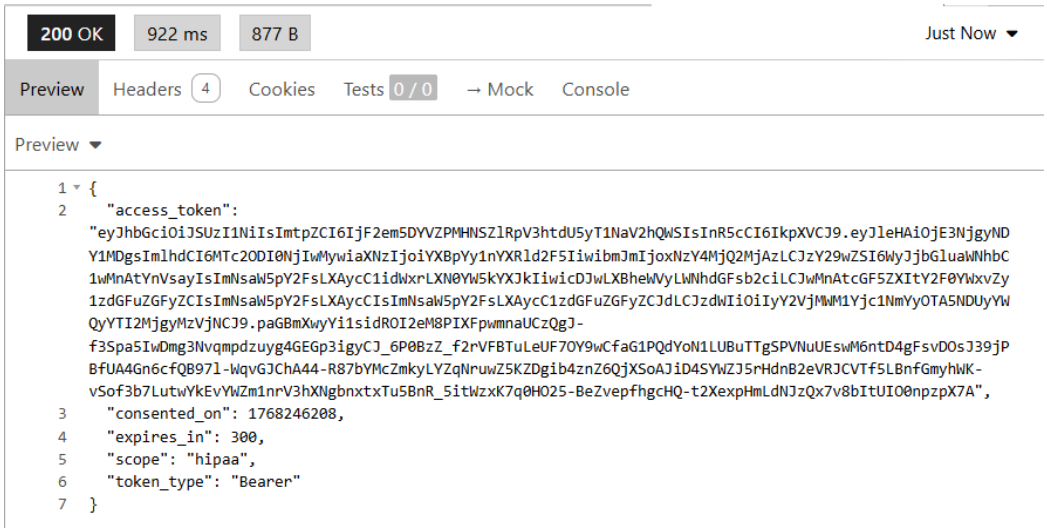
Populate the Client ID and Secret

In the **Body > Form** tab, enter the unique values for **client_id** and **client_secret** provided by Availity. If you are using variables as recommended, enter the variable names as shown below, and then click **Send**.



Response to the POST /token Call

The response to this request returns an **access_token** for use in subsequent API requests (see line 3 in the screenshot below). The value between the quotation marks is the token.



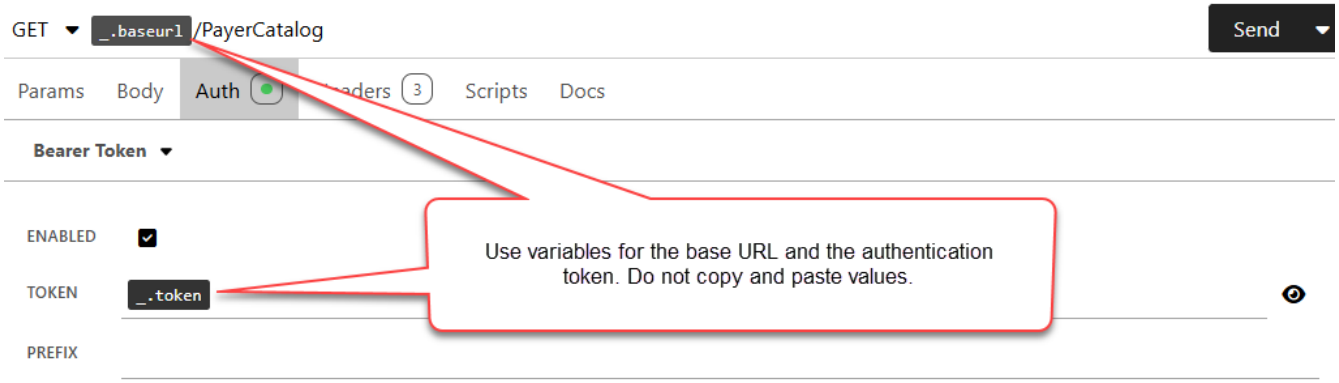
For all remaining API requests, include the access_token in the header named Authorization using the format Bearer <access_token>. If you have defined a variable for access_token, you can avoid copying and pasting the value by setting the Bearer Token field to _. token, which retrieves the value from the variable.

2. GET /PayerCatalog

Availity provides a Payer Catalog to help locate details for the Old Plan. The catalog includes payerId, payerName, and payerDescription. If a payer offers multiple plans, the catalog also lists planId, planName, and planDescription for each plan.

This information is required for subsequent calls to the Payer-to-Payer Connectivity Hub. To retrieve a patient's data from their Old Plan, you must know the Old Payer (to populate the CoverageToMatch section of the $member-match request) and the PatientID so the Hub can request the correct patient's data.

After you obtain the **access_token**, send a **GET** /PayerCatalog request as shown below, and include the **Token** parameter.



The response is a JSON-formatted array listing all payers participating in the Payer-to-Payer Connectivity Hub and the plans each offers, as shown in the following example:

200 OK | 423 ms | 1768 B | Just Now

Preview | Headers 13 | Cookies | Tests 0 / 0 | → Mock | Console

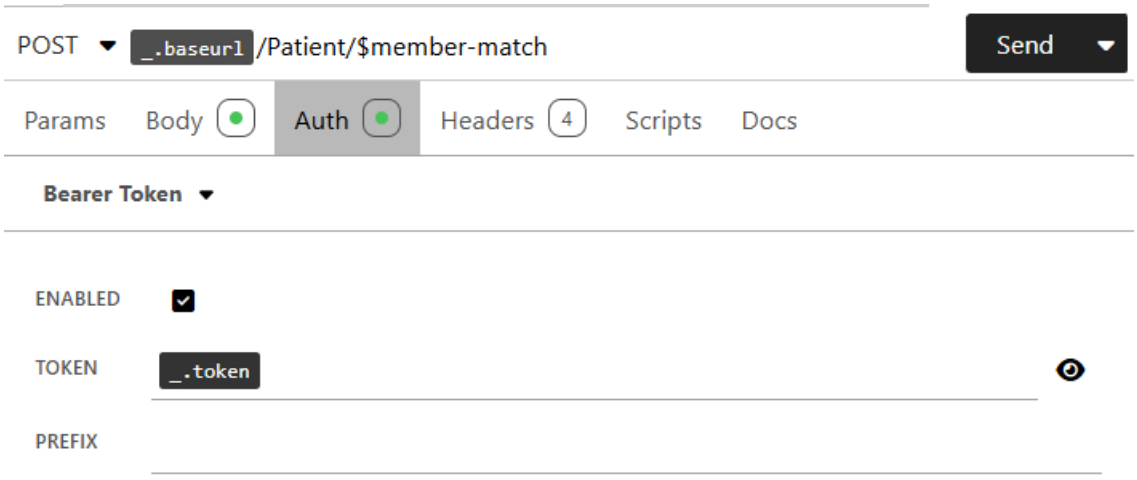
Preview


```
{
  "payers": [
    {
      "payerId": "hcsc",
      "payerName": "HCSC",
      "payerDescription": "HCSC Description",
      "isPayerCustomer": null,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "floridablue",
      "payerName": "Florida Blue",
      "payerDescription": "Florida Blue Description",
      "isPayerCustomer": null,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "centene-bulk-test",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "centene-bulk",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "humana",
      "payerName": "Humana",
      "payerDescription": "Humana Description",
      "isPayerCustomer": true,
      "isPayerConnected": true,
      "plans": []
    },
    {
      "payerId": "centene",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
```

3. POST /Patient/$member-match

Send a **POST** request to the /Patient/$member-match endpoint. This request retrieves the FHIR Patient ID for the individual from the old payer's system.

In the **Bearer** tab, ensure that the **TOKEN** field is populated with the variable name used in previous requests.



Click the **Body** tab. The request body includes a JSON-formatted payload with required parameters, such as:

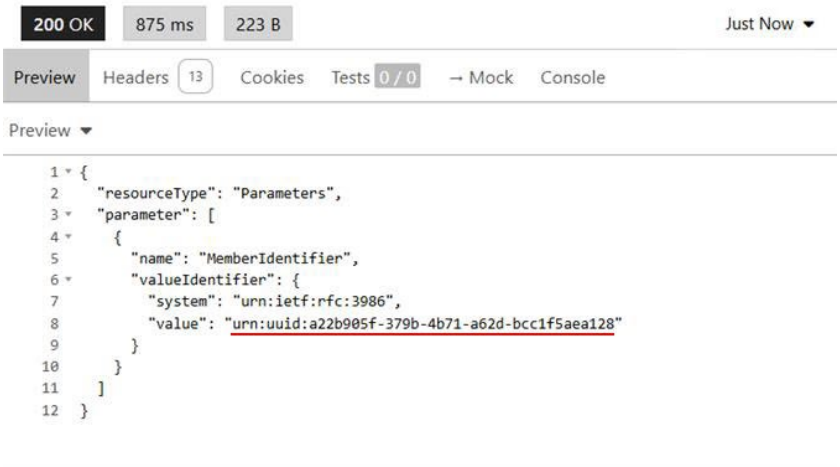
1. **Member information** – Includes the patient's identifier from the old payer's system.
2. **Old payer details** – Identifies who to request the patient's data from (**CoverageToMatch**).
3. **New payer details** – Identifies who will receive the patient's data (**CoverageToLink**).
4. **Patient consent** – Specifies which data the old payer can release to the new payer (not shown here).

The **Body** tab contains a JSON-formatted array of parameters. Sample arrays are available on the Availity Clinical Solutions Support Hub click here.

NOTE	You must be a registered user to access the Support Hub. Contact your Availity representative for assistance.

Response to the POST /Patient/$member-match Call

The response for this call appears as follows:



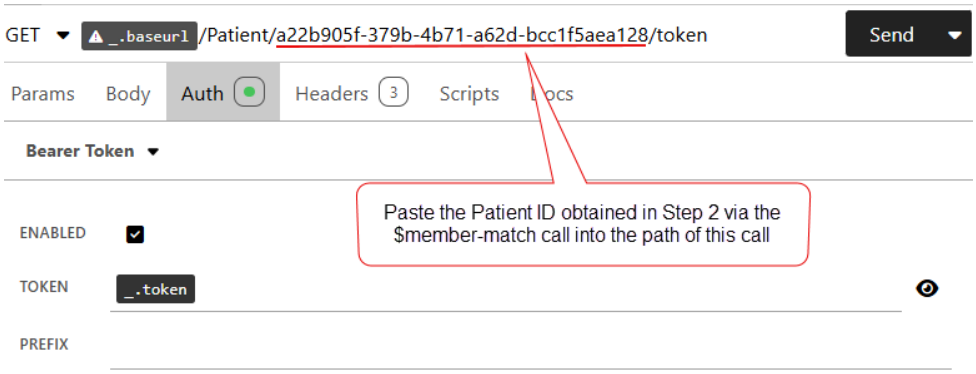
Line 8 of the response contains the individual's FHIR Patient resource ID on the Old Payer's system, highlighted in red. Use this FHIR Patient ID in the final two API requests.

NOTE	The Payer-to-Payer Connectivity Hub acts as a proxy and maps data between the two systems, preventing each party from viewing information in the other party's system. Thus, these values are *not* the literal FHIR Patient ID value on the Old Payer, but an Availity-provided value that the Connectivity Hub *maps* to the Old Payer's FHIR Patient ID, a temporary ID to facilitate the secure exchange of data. A similar function is performed with the response to the GET Patient/$everything call described in Step 5, where the Connectivity Hub substitutes Availity URLs in the pagination links. Data is requested from Availity, not directly from the Old Payer.

4. GET /Patient/{id}/token

So far, authorization tokens have granted permission to interact with the Payer-to-Payer Connectivity Hub. However, an additional layer of authentication is required to access data for a specific individual. This section explains how to obtain the patient access token. In the fifth and final API request, you will use this patient access token along with your general authorization token to request resources related to the specific patient.

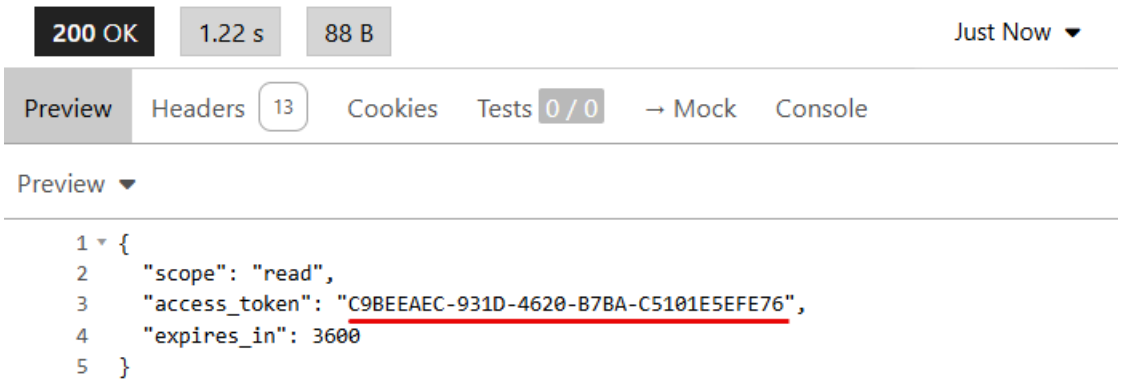
Insert the Patient ID obtained in Step 3 as a path variable in the API request, between the /Patient parameter and the /token parameter:



This request indicates that the sender needs a token to access Patient X's data.

Response to the GET /Patient/{id}/token Call

The response to this request is shown below. The critical component is the patient **access_token** value on line 3, highlighted in red. This patient-level access token grants permission to request the individual's FHIR Patient resource on the Old Payer's server. Use this access_token in the next and final API request to retrieve all resources for the patient.



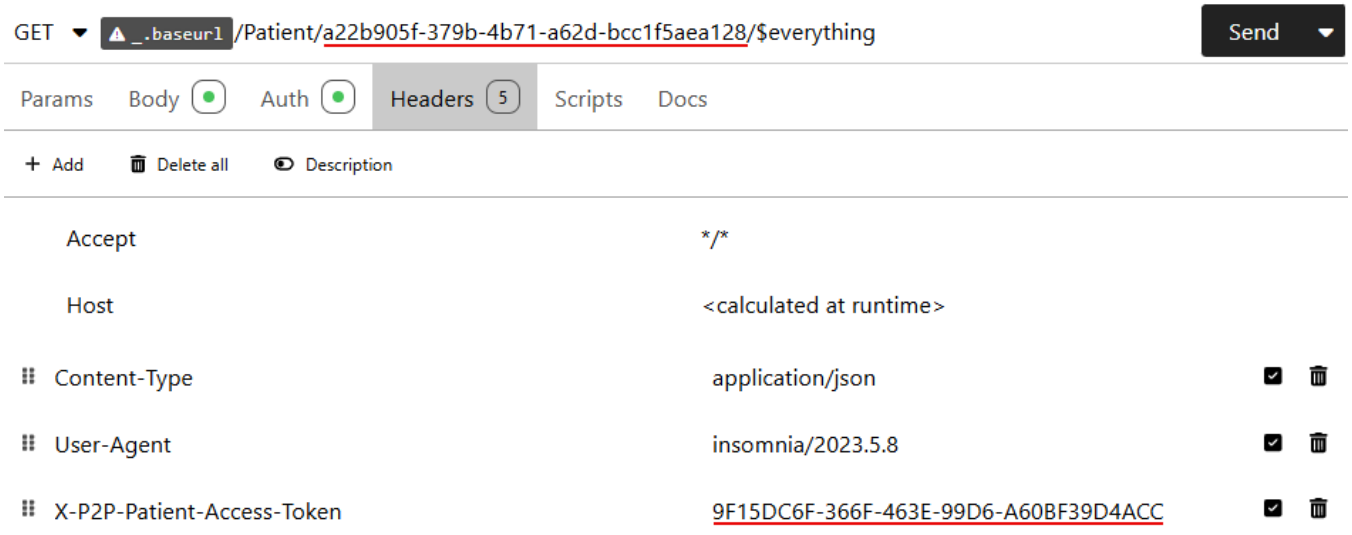
5. GET /Patient/{id}/$everything

Finally, send a GET /Patient/{id}/$everything request to retrieve the patient's data from the Old Payer.

Insert the Patient ID as a path variable, just as in Step 4.

In the Headers tab, add a custom header named **X-P2P-Patient-Access-Token**, as shown below. Give it the value of the Patient **access_token** obtained in Step 4.

The receiving system uses this field to validate that the incoming request includes the required authentication to access the patient's data. The usual Bearer token in the **Auth** tab is also required, as in Steps 2, 3, and 4.



After populating these three values – **Auth Token**, **Patient ID** in the path, and **X-P2P-Patient-Access-Token** – select **Send**.

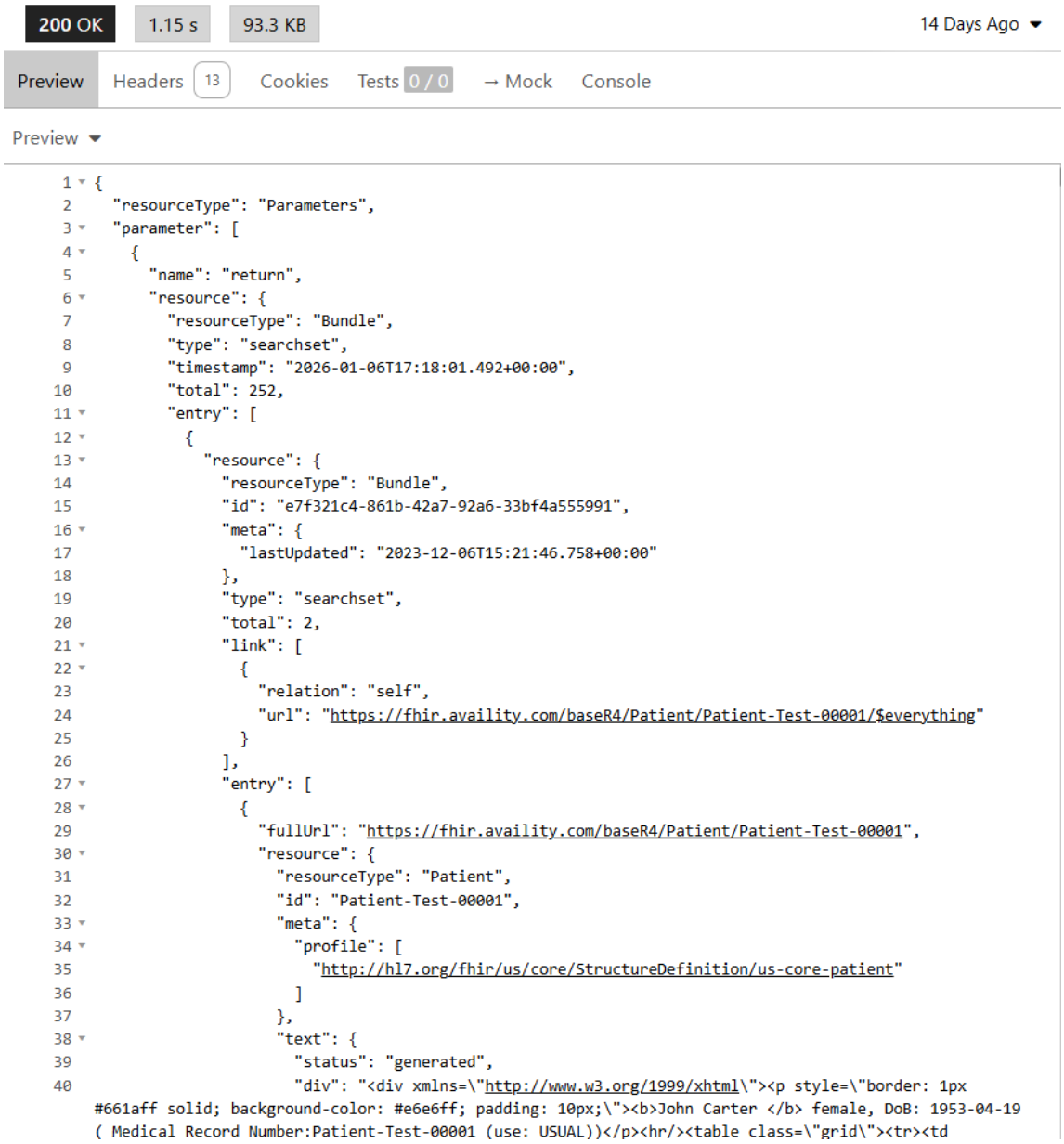
Response to the GET /Patient/{id}/$everything Call

The response to the GET /Patient/{id}/$everything request is a FHIR Bundle that contains all data shared by the Old Payer in accordance with the terms of the consent array.

NOTE	The Mock Payer Service does not evaluate the consent array.

A brief example is shown below. An actual FHIR bundle for a Patient will contain far more data. The sample shown has only the **self** link (lines 16-17); typically, there are **previous** and **next** links provided in the **link** array. FHIR provides these pagination links to make navigating a large response easier.

NOTE	The Connectivity Hub substitutes (proxies) Availity URLs in the pagination links to avoid exposing the Old Payer's details.




```
{
  "resourceType": "Parameters",
  "parameter": [
    {
      "name": "return",
      "resource": {
        "resourceType": "Bundle",
        "type": "searchset",
        "timestamp": "2026-01-06T17:18:01.492+00:00",
        "total": 252,
        "entry": [
          {
            "resource": {
              "resourceType": "Bundle",
              "id": "e7f321c4-861b-42a7-92a6-33bf4a555991",
              "meta": {
                "lastUpdated": "2023-12-06T15:21:46.758+00:00"
              },
              "type": "searchset",
              "total": 2,
              "link": [
                {
                  "relation": "self",
                  "url": "https://fhir.availity.com/baseR4/Patient/Patient-Test-00001/$everything"
                }
              ],
              "entry": [
                {
                  "fullUrl": "https://fhir.availity.com/baseR4/Patient/Patient-Test-00001",
                  "resource": {
                    "resourceType": "Patient",
                    "id": "Patient-Test-00001",
                    "meta": {
                      "profile": [
                        "http://hl7.org/fhir/us/core/StructureDefinition/us-core-patient"
                      ]
                    },
                    "text": {
                      "status": "generated",
                      "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><p style=\"border: 1px
#661aff solid; background-color: #e6e6ff; padding: 10px;\"><b>John Carter </b> female, DoB: 1953-04-19
( Medical Record Number:Patient-Test-00001 (use: USUAL))</p><hr/><table class=\"grid\"><tr><td
```


NOTE	The data provided by the Old Payer depends on the details of the Consent Request as specified by the member. For example, the member may give their New Payer consent to retrieve only data within a particular date range or exclude sensitive information such as mental health data. Refer to HL7's Davinci specification for further details.

Example: Member Match Request

The JSON payload of the **GET Patient/$member-match** request contains four key arrays of data. In the API call overview, we showed those arrays collapsed for concise presentation. It is worth taking a moment to expand those arrays and examine the contents to better understand what each section is and which parameters are required. Availity's Payer-to-Payer data exchange implementation hews closely to the HL7 Implementation Guide, so we are showing the full picture that the IG presents. However, for the specific use case we are addressing, only a few of the many parameters are of note initially, such as patient identifier, the Old Payer (Coverage-to-Match), the New Payer (Coverage-to-Link), and the data disclosure boundaries (Consent).

As our solution matures, additional pieces may become necessary to provide as input to the Payer-to-Payer Connectivity Hub when you are acting as the New Payer or to include in response to a request for data when you are acting as the Old Payer. For example, in a production environment, we anticipate that a New Payer requesting data will provide an accurately-populated Consent array defining exactly what data the patient has agreed to release, and on the other side of the exchange, that partners acting as Old Payers will respect the terms of the Consent array, limiting the contents of their response to the GET $everything call to stay within the Patient's consented boundaries.

For the time being, If you are using the API collection that Availity provides, testing with the pre-configured patient, the Patient identifier value will already be populated, so all you'll need to do is send the call and provide the values for access_token and (in the $everything call) the Patient access token.

The four arrays in the JSON body of the $member-match operation include:

1. Patient - this represents the individual's Old Payer member ID.
2. CoverageToMatch - this represents the Old Payer system to query.
3. CoverageToLink - this represents the New Payer system requesting the data.
4. Consent - this details the date range, information permitted, and other details of what the patient has consented to have transmitted to the New Payer.

The following sections describe each of the four sections in the request body:

Patient

The Patient section represents the identity of the individual so that the Old Payer can locate their data. In the example below, the New Payer has provided the individual's member ID on the Old Payer's system as well as their name and birth date as additional identity markers.

```
{
"name": "MemberPatient",
"resource": {
  "resourceType": "Patient",
  "identifier": [
```

```
    {
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/v2-0203",
            "code": "MB"
          }
        ]
      },
      "system": "http://example.org/old-payer/identifiers/member",
      "value": "F16BF601-6125-467F-8617-E8F255DB2939",
      "assigner": {
        "display": "Old Payer"
      }
    }
  ],
  "name": [
    {
      "use": "official",
      "family": "Washburne",
      "given": [
        "Zoe"
      ]
    }
  ],
  "gender": "female",
  "birthDate": "1987-02-20"
  }
}
```

CoverageToMatch

Coverage to Match represents the Old Payer from whom the patient's data is being requested. The critical values here include the Payor Identifier values.


```
{
"name": "CoverageToMatch",
"resource": {
  "resourceType": "Coverage",
  "id": "9876B1",
  "identifier": [
    {
      "system": "http://example.org/old-payer",
      "value": "asdf"
    }
  ],
  "status": "draft",
  "beneficiary": {
    "reference": "Patient/13460"
  },
  "period": {
    "start": "2011-05-23",
    "end": "2012-05-23"
  },
  "payor": [
    {
      "identifier": {
        "system": "https://centene.org/fhir/Patient/$member-match",
        "value": "centene-mock"
      },
      "display": "Old Health Plan"
    }
  ],
  "class": [
    {
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/coverage-class",
            "code": "group"
          }
        ]
      },
      "value": "CB135"
    },
    {
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/coverage-class",
            "code": "plan"
```

```
          }
        ]
      },
      "value": "B37FC"
    },
    {
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/coverage-class",
            "code": "subplan"
          }
        ]
      },
      "value": "P7"
    },
    {
      "type": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/coverage-class",
            "code": "class"
          }
        ]
      },
      "value": "SILVER"
    }
  ]
}
```

CoverageToLink

Coverage to Link represents the New Payer who is requesting the patient's data.


```
{
"name": "CoverageToLink",
"resource": {
  "resourceType": "Coverage",
  "id": "AA87654",
  "identifier": [
    {
      "system": "http://example.org/new-payer/identifiers/coverage",
      "value": "234567"
    }
  ],
  "status": "active",
  "beneficiary": {
    "reference": "Patient/13460"
  },
  "payor": [
    {
      "identifier": {
        "system": "http://hl7.org/fhir/sid/us-npi",
        "value": "0123456789"
      },
      "display": "New Health Plan"
    }
  ]
}
```


Consent

The Consent section details what data the patient agrees to allow their Old Plan to share with their New Plan. This may include constraints such as a data range, the nature of the information permitted to be shared, and other details. Two common examples that someone might choose not to share are data related to addiction services and mental health data.


```
{
"name": "Consent",
"resource": {
  "resourceType": "Consent",
  "status": "active",
  "scope": {
    "coding": [
      {
        "system": "http://terminology.hl7.org/CodeSystem/consentscope",
        "code": "patient-privacy"
      }
    ]
```

```
},
"category": [
  {
    "coding": [
      {
        "system": "http://terminology.hl7.org/CodeSystem/v3-ActCode",
        "code": "IDSCL"
      }
    ]
  }
],
"patient": {
  "reference": "Patient/13460"
},
"performer": [
  {
    "reference": "Patient/13460"
  }
],
"sourceReference": {
  "reference": "http://example.org/DocumentReference/someconsent"
},
"policy": [
  {
    "uri": "http://hl7.org/fhir/us/davinci-hrex/StructureDefinition-hrex-consent.html#regular"
  }
],
"provision": {
  "type": "permit",
  "period": {
    "start": "2022-01-01",
    "end": "2022-01-31"
  },
  "actor": [
    {
      "role": {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/provenance-participant-type",
            "code": "performer"
          }
        ]
      },
      "reference": {
        "identifier": {
          "system": "http://hl7.org/fhir/sid/us-npi",
          "value": "9876543210"
        },
        "display": "Old Health Plan"
      }
    },
    {
```

```
        "role": {
          "coding": [
            {
              "system": "http://terminology.hl7.org/CodeSystem/v3-ParticipationType",
              "code": "IRCP"
            }
          ]
        },
        "reference": {
          "identifier": {
            "system": "http://hl7.org/fhir/sid/us-npi",
            "value": "0123456789"
          },
          "display": "New Health Plan"
        }
      }
    ],
    "action": [
      {
        "coding": [
          {
            "system": "http://terminology.hl7.org/CodeSystem/consentaction",
            "code": "disclose"
          }
        ]
      }
    ]
  }
}
```

Payer-to-Payer – Bulk API Call Sequence

Overview

Retrieving bulk patient data from the Old Payer requires a sequence of API requests:

1. POST /token to obtain an access token.
2. GET /PayerCatalog to identify the Old Payer ID for routing the patient data request to the correct party.
3. POST /fhir/r4/Group/$multi-member-match to match multiple members in the Old Payer's system.
4. GET /fhir/r4/Group/{id}/token to obtain a token scoped to the group of members.
5. GET /fhir/r4/Group/{id}/$davinci-data-export to initiate export of member data in NDJSON format.
6. GET /fhir/r4/Group/$davinci-data-export/{id}/status to check the status of the bulk export request.

Each step is described in more detail below with examples shown in the Insomnia API tool. If you are using Postman rather than Insomnia, there will be some minor variations, such as screen layout, configuring environments, and the naming convention for variables, but the general process and sequence of API calls remains the same.

NOTE 1	The examples provided here assume that you are using the Insomnia API collection that Availity provides. This collection includes pre-configured API calls for various mock patients, pre-populated with the URL for our API gateway (https://qua.api.availity.com/). Nonetheless, please pay attention to the URL path in the screenshots, as the first API call includes "v1/" after the URL, and the remaining three calls include "fhir/r4/" after the URL. We will define a variable for **baseUrl** to make testing a little easier.

NOTE 2	For security, Availity's API gateway limits the duration of authentication tokens to 300 seconds (5 minutes). You will need to complete the entire sequence of calls within this timeframe before your access token expires.

1. POST /token

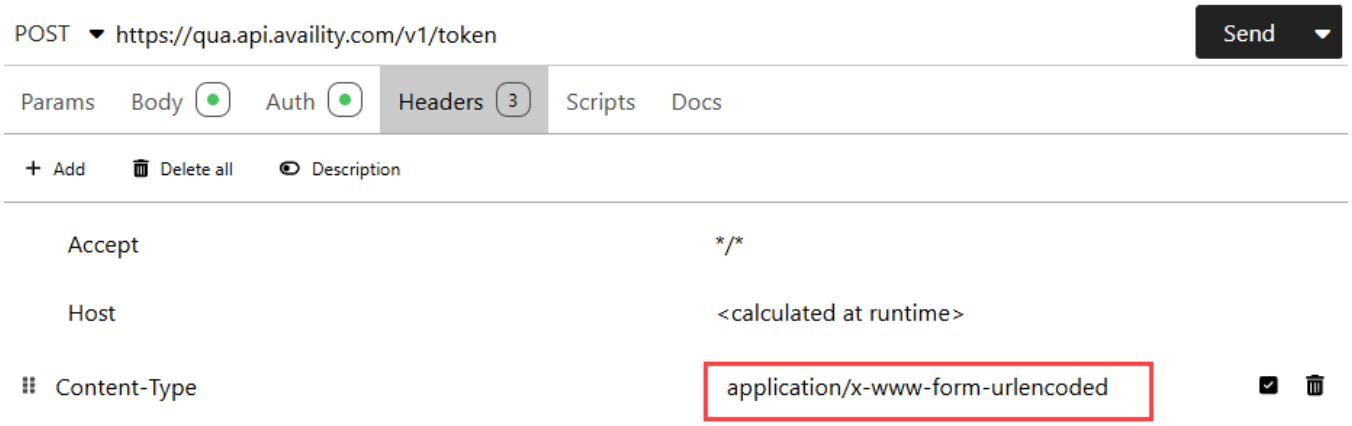
This step authenticates the Payer-to-Payer Connectivity Hub and retrieves the authorization token required for subsequent requests. Before you send this request:

1. Note the URL for this request: https://qua.api.availity.com/v1. All subsequent requests use the same URL defined in the **baseUrl** variable.
2. Set the body type to **Form**.
3. Enter the form data values for **ClientID** and **client_secret.**

Set the Body Type to **Form**

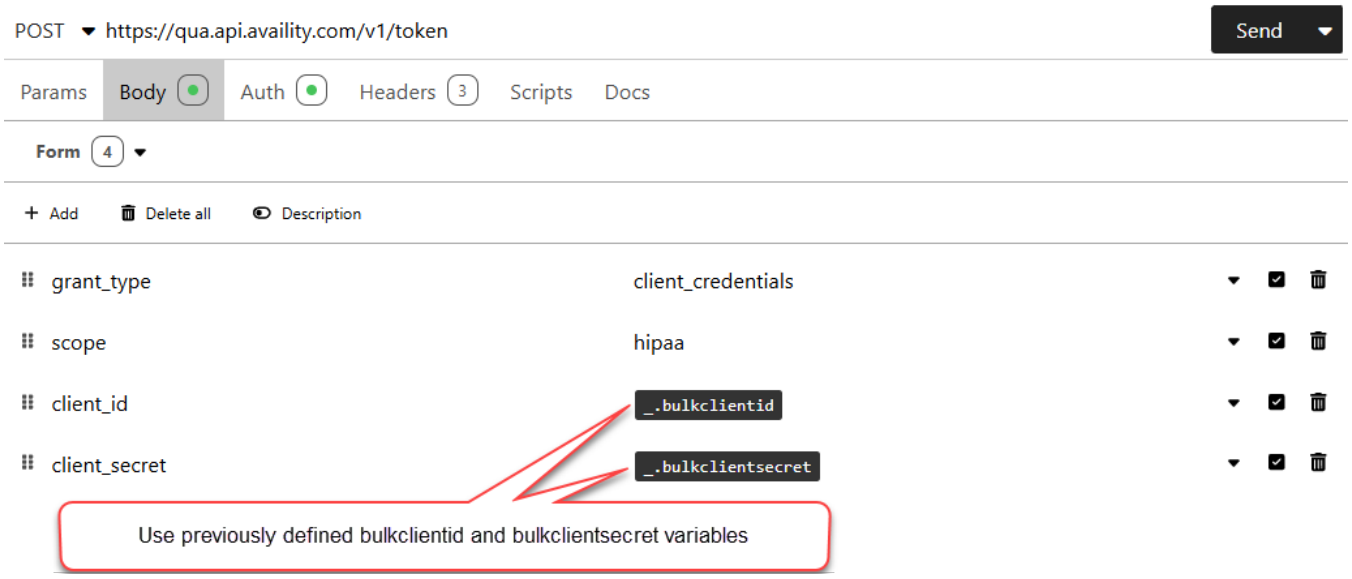
NOTE	This is already done for you in the P2P API collection Availity make available, but it is worth verifying if only to familiarize yourself with the different tabs and components of an API call as represented in Insomnia. Postman's UI is similar.

The POST /token call uses a Form data Body, while the rest of the API calls have JSON payloads in their Body. In an API tool such as Postman or Insomnia, when you change the body type pulldown from **JSON** to **Form**, the Headers tab will be populated with a **Content-Type** header having a value of **application/x-www-form-urlencoded**, as shown below:



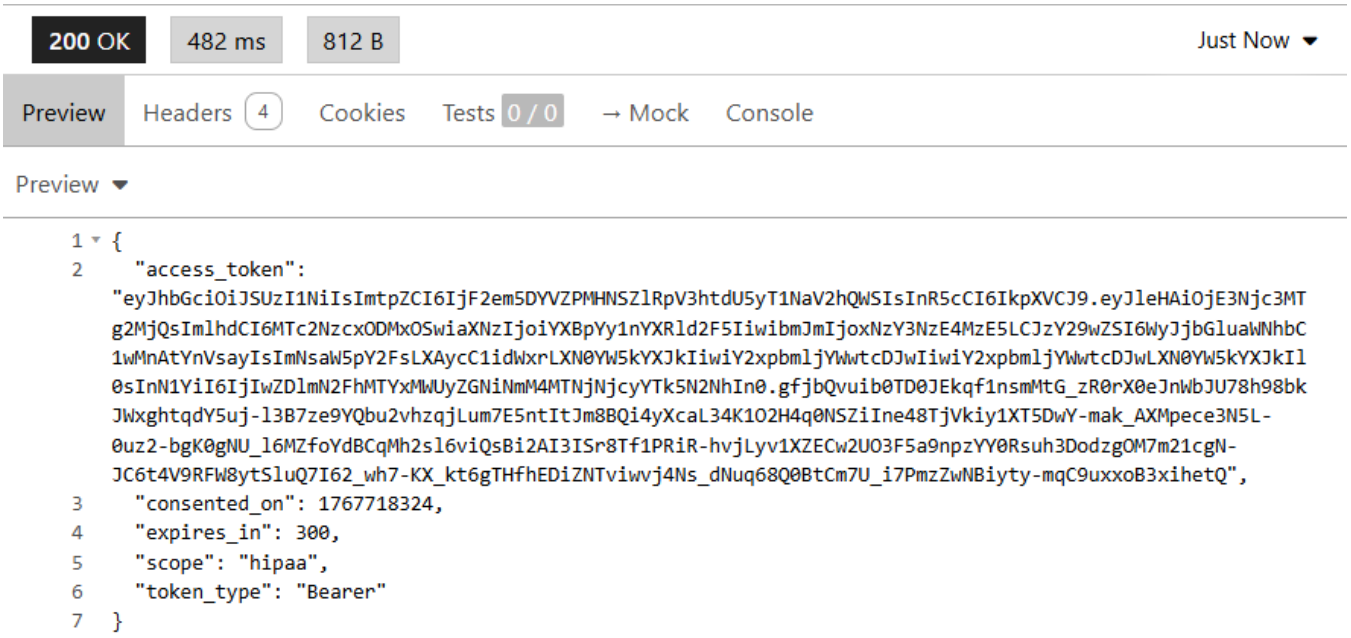
Populate the Client ID and Secret

In the **Body > Form** tab, populate **client_id** and **client_secret** with the unique values Availity provided to you. If you are using variables as suggested, populate the form with the variable names as shown below, and click **Send**.



Response to the POST /token Call

This call returns an **access_token** that you use for the remaining API calls (refer to the following screenshot). The value enclosed in quotation marks is the token.



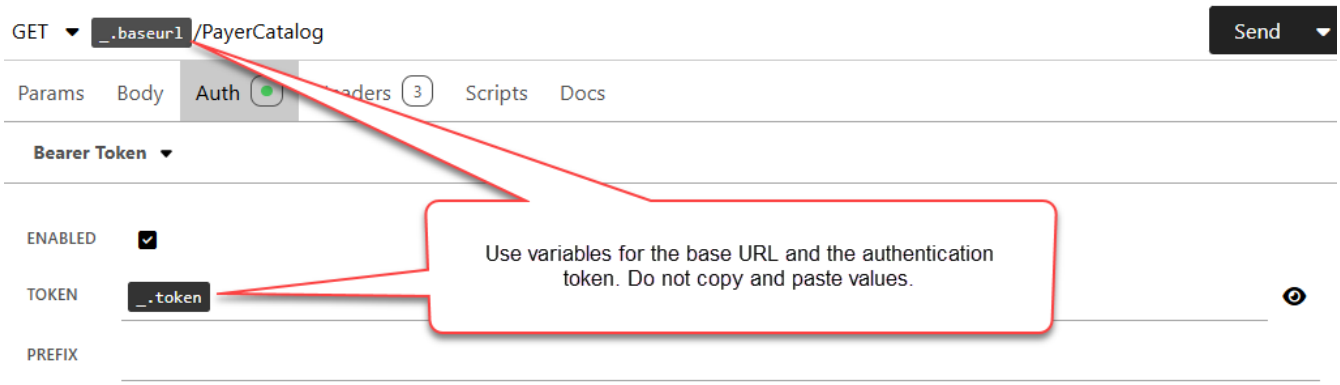
In the remaining API calls you will place the access_token into the header value named Authorization in the format Bearer <access_token>. BUT, if you have set up a variable for access_token, you can avoid copying/pasting this value repeatedly by populating the value for Bearer Token as "_.token", to take its value from the variable we defined.

2. GET /PayerCatalog

Availity provides a Payer Catalog to help identify details for the Old Plan. The catalog includes payerId, payerName, and payerDescription. If a payer offers multiple plans, the catalog also lists planId, planName, and planDescription for each plan.

This information is required for subsequent requests to the Payer-to-Payer Connectivity Hub. To retrieve a patient's data from their Old Plan, you must know the Old Payer (to populate the CoverageToMatch section of the $member-match request) and the PatientID so the Hub can request the correct patient's data.

After you obtain the access_token, send a **GET** /PayerCatalog request as shown below, and include the **Token** parameter.



The response is a JSON-formatted array that lists all payers participating in the Payer-to-Payer Connectivity Hub and the plans each offers, as shown in the following example:

200 OK | 423 ms | 1768 B | Just Now

Preview | Headers 13 | Cookies | Tests 0 / 0 | → Mock | Console

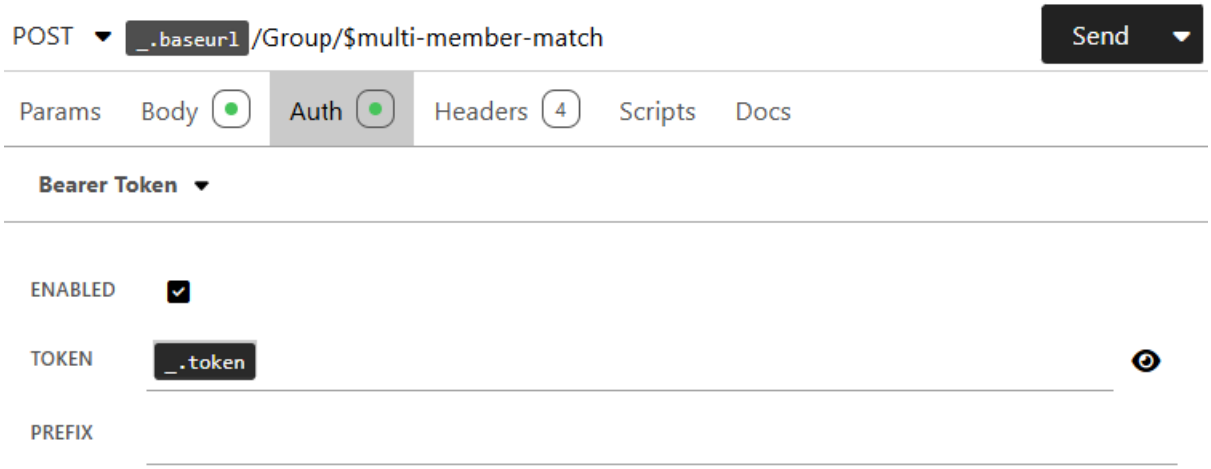
Preview


```
{
  "payers": [
    {
      "payerId": "hcsc",
      "payerName": "HCSC",
      "payerDescription": "HCSC Description",
      "isPayerCustomer": null,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "floridablue",
      "payerName": "Florida Blue",
      "payerDescription": "Florida Blue Description",
      "isPayerCustomer": null,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "centene-bulk-test",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "centene-bulk",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
      "payerId": "humana",
      "payerName": "Humana",
      "payerDescription": "Humana Description",
      "isPayerCustomer": true,
      "isPayerConnected": true,
      "plans": []
    },
    {
      "payerId": "centene",
      "payerName": "Test Payer",
      "payerDescription": "Test Description",
      "isPayerCustomer": true,
      "isPayerConnected": null,
      "plans": []
    },
    {
```

3. POST /fhir/r4/Group/$multi-member-match

Send a **POST** request to the /fhir/r4/Group/$multi-member-match endpoint. This request retrieves the FHIR Patient ID values for multiple patients from the old payer's system.

In the **Bearer Token** tab, ensure that the **TOKEN** field is populated with the variable name used in previous calls.



Click the **Body** tab. The request body includes a JSON-formatted payload with required parameters, such as:

1. **Member information** – Includes the patient's identifier from the old payer's system.
2. **Old payer details** – Identifies who to request the patient's data from (**CoverageToMatch**).
3. **New payer details** – Identifies who will receive the patient's data (**CoverageToLink**).
4. **Patient consent** – Specifies which data the old payer can release to the new payer (not shown here).

The **Body** tab contains a JSON-formatted array of parameters. Sample arrays are available on the Availity Clinical Solutions Support Hub click here.

NOTE	You must be a registered user to access the Support Hub. Contact your Availity representative for assistance.

Response to the POST /fhir/r4/Group/$multi-member-match

The response to this call includes three sections:

- ResourceIdentifier – This section includes a group of matched patients.
- NoMatch – This section includes a group of unmatched patients.
- Consent Constraint – This section has a group of patients who has consent constraints.

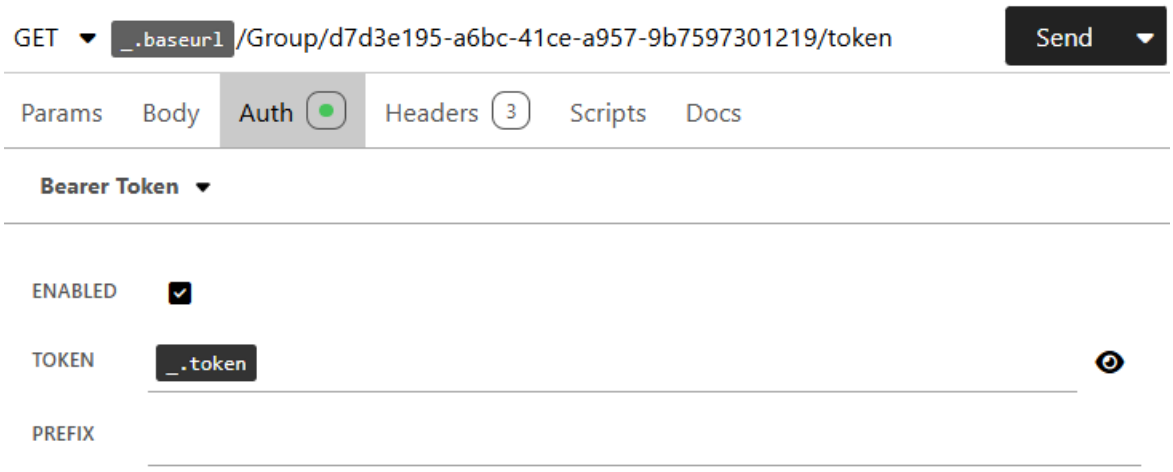
To view the detailed sample response, click here. to view sample response.

NOTE	The Payer-to-Payer Connectivity Hub acts as a proxy, blinding each party to information on the other's system by mapping between them. Thus, these values are *not* the literal FHIR Patient ID value on the Old Payer, but an Availity-provided value that the Connectivity Hub *maps* to the Old Payer's FHIR Patient ID, a temporary ID to facilitate the secure exchange of data.

4. GET /fhir/r4/Group/{id}/token

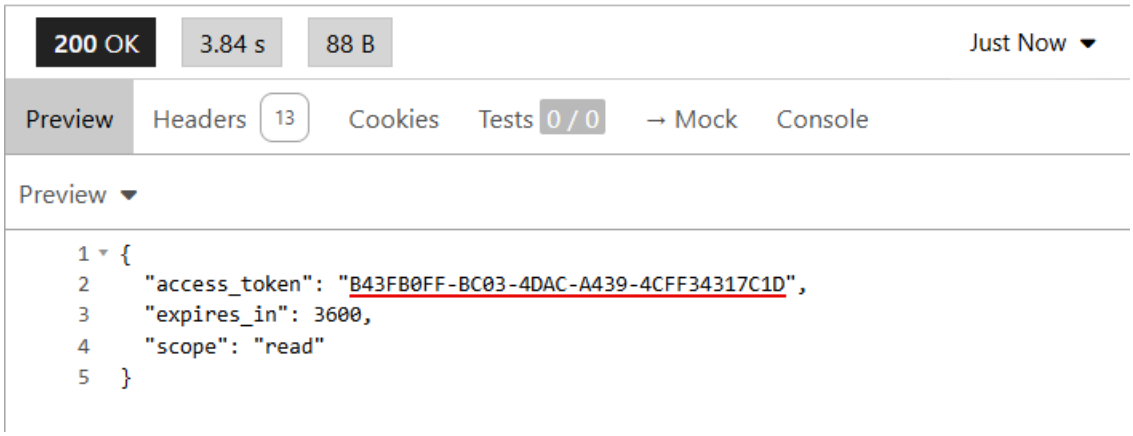
Use your general authorization token to access the Payer-to-Payer Connectivity Hub. To access data for multiple patients in a group, you need an additional layer of authentication. This section explains how to obtain the **Group Access Token**. In the fifth API request, use the Group Access Token along with your general authorization token to request resources for all patients in the group.

The Group ID from Step 3 should be included directly in the API call as a **path variable**, placed between the /Group segment and the /token segment.



Response to the GET /fhir/r4/Group/{id}/token Call

The response to this request includes the Group Access Token, shown on line 2 in the example below. This token grants permission to request the FHIR Patient resource for multiple patients on the old payer's server. Use the Group Access Token value in the next API call to retrieve all patient resources that belong to the group.



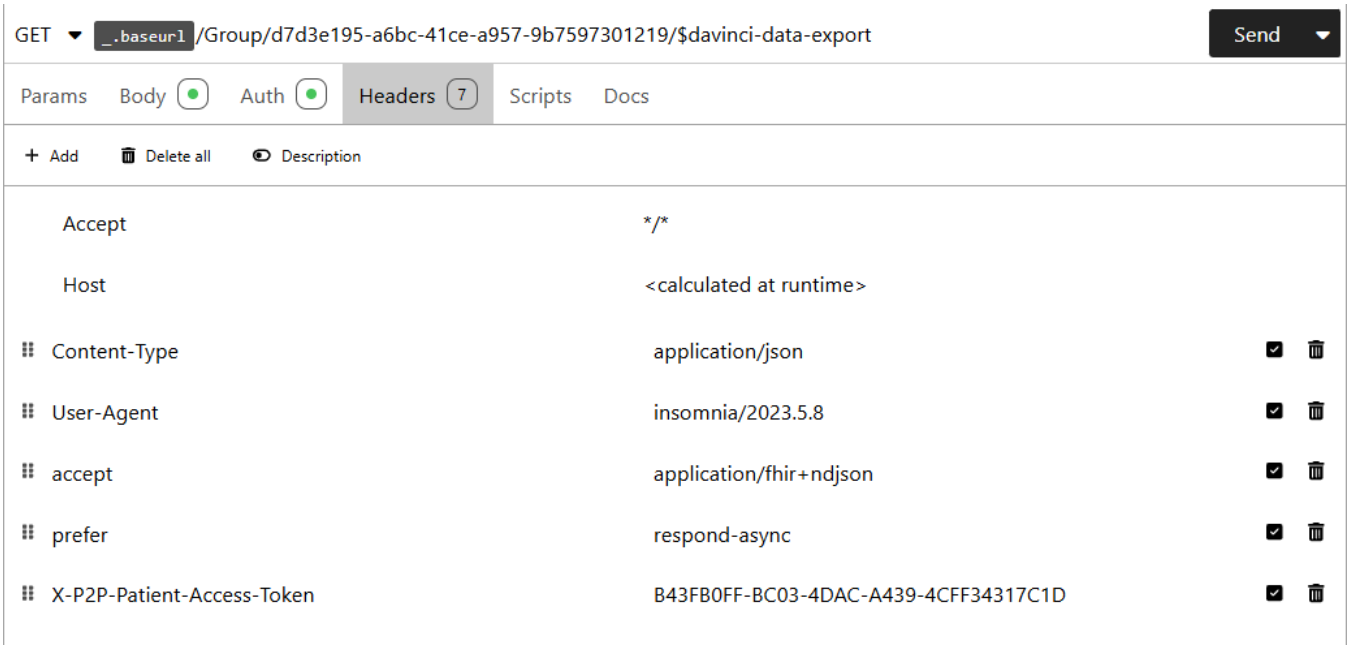
5. GET /fhir/r4/Group/{id}/$davinci-data-export

Use the GET /fhir/r4/Group/{id}/$davinci-data-export call to retrieve the multiple patient data from the Old Payer.

Insert the Group ID as a path variable, just as in Step 4.

In the Headers tab, add a custom header called **X-P2P-Patient-Access-Token** and set its value to the **Group access_token** from Step 4.

The receiving system will use this field to validate that the incoming request has the necessary authentication to obtain the multiple Patient data. (The usual Bearer Token in the Auth tab is also still required, as in Steps 2, 3, and 4.)



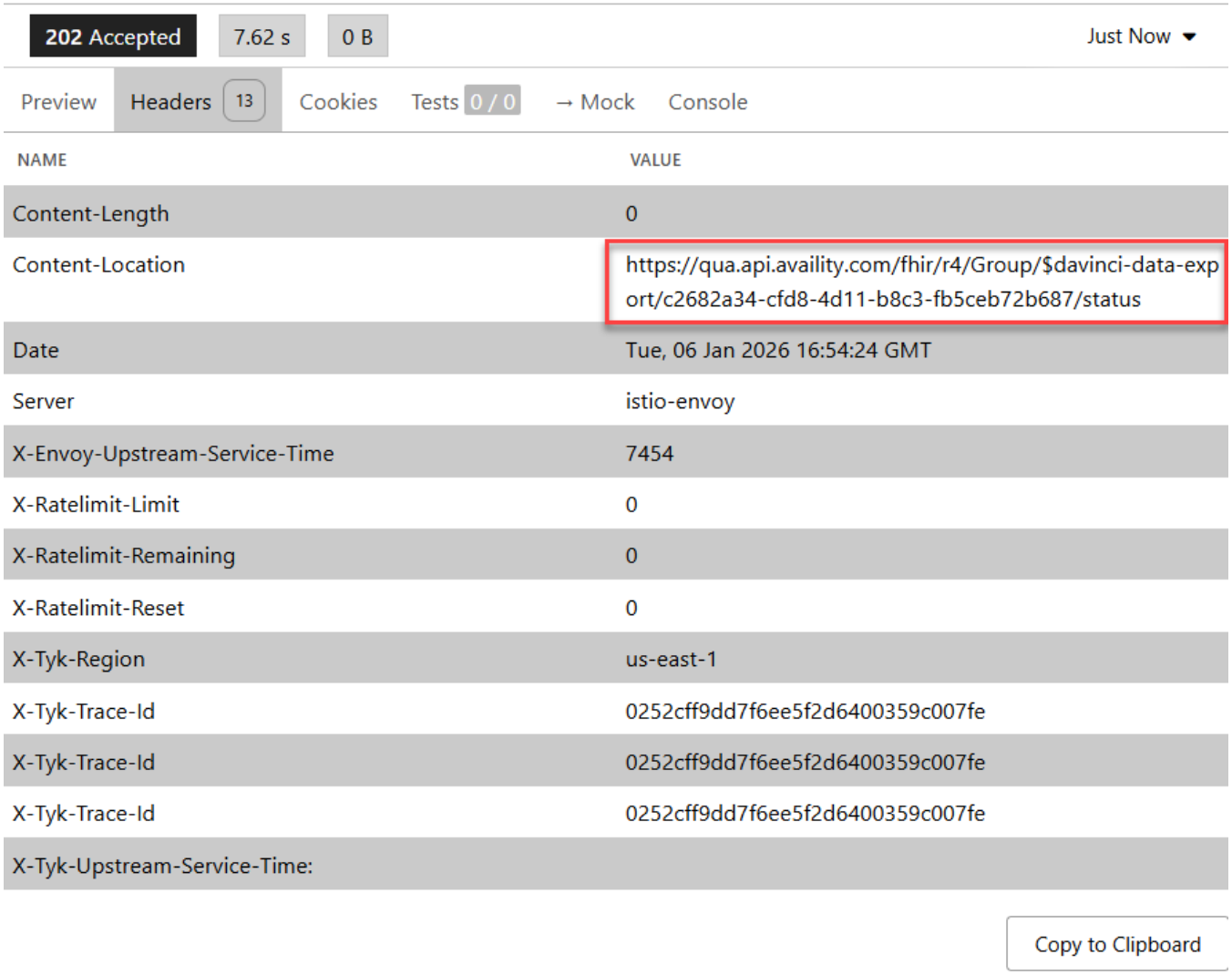
After populating these three values – **Auth Token**, **Group ID** in the path, and **X-P2P-Patient-Access-Token** – select **Send**.

Response to the GET /fhir/r4/Group/{id}/$davinci-data-export Call

The response to the **GET** /fhir/r4/Group/{id}/$davinci-data-export call is a FHIR Bundle containing all the data shared by the Old Payer in accordance with the terms of the Consent array.

> **NOTE** The Mock Payer Service does not evaluate the Consent array.

Use the Content location URL in the Headers tab to download the data,



NOTE	The data provided by the Old Payer depends on the details of the Consent Request as specified by the member. For example, the member may give their New Payer consent to retrieve only data within a particular date range or exclude sensitive information such as mental health data. Refer to HL7's Davinci specification for further details.

6. GET /fhir/r4/Group/$davinci-data-export/{id}/status

Use the GET /fhir/r4/Group/$davinci-data-export/{id}/status call to retrieve the URL for obtaining multiple patients' data from the Old Payer.

Insert the Group ID as a path variable, just as in Step 4.

In the Headers tab, add a custom header named **X-P2P-Patient-Access-Token**, as shown below. Give it the value of the **Group access_token** obtained in Step 4.

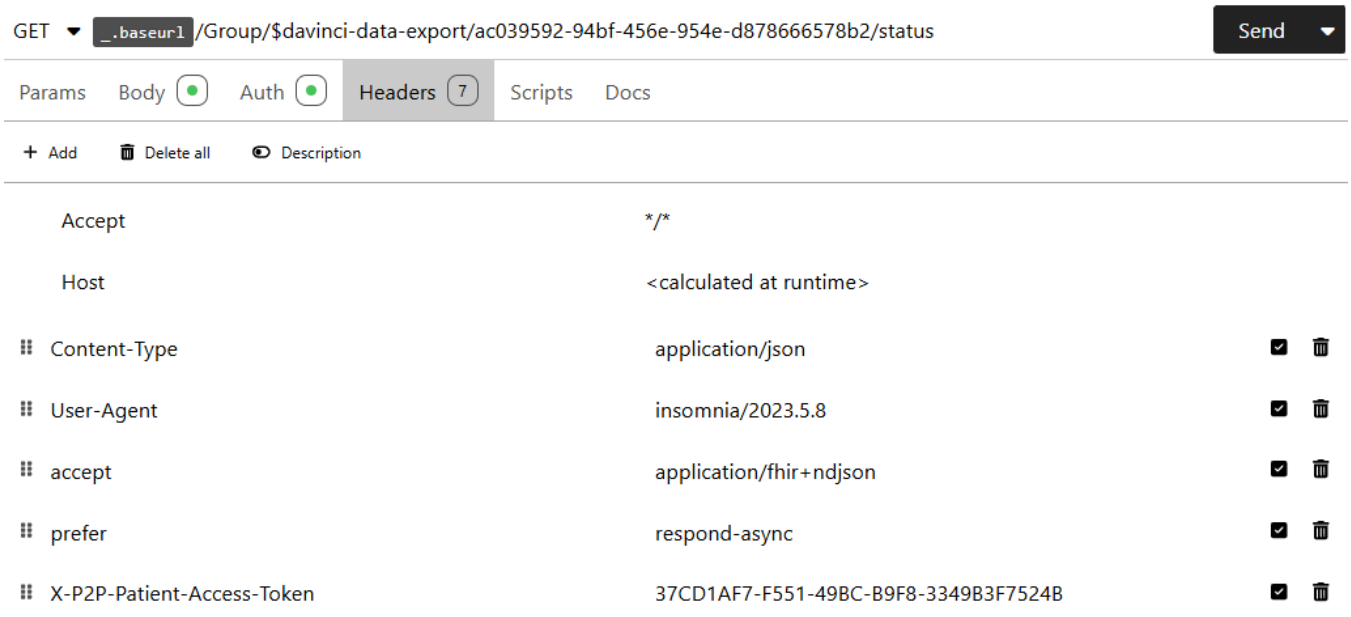
The receiving system will use this field to validate that the incoming request has the necessary authentication to obtain the multiple patients' data. The request returns a status endpoint URL for tracking the file generation progress.

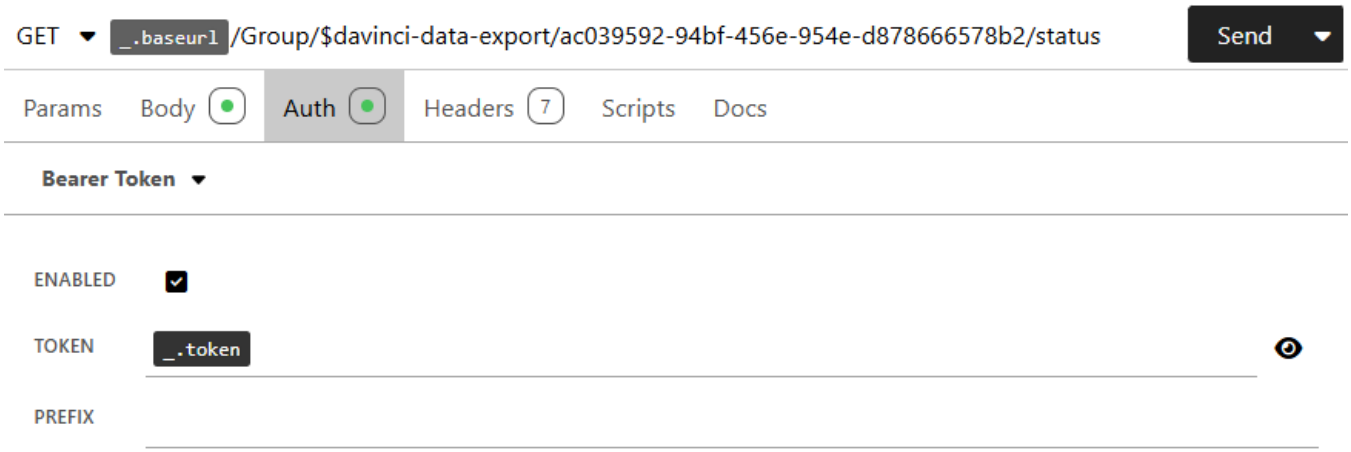
Status URLs will be requested many times using the HL7 FHIR Asynchronous Request Pattern (https://hl7.org/fhir/R4/async.html)

- **Polling with Exponential Backoff**
    - Instead of constant polling, increase the delay between status checks exponentially (e.g., 1s → 2s → 4s → 8s).
    - This prevents overwhelming the server and complies with HL7 guidance.
- **Honor Retry-After Header**
    - If the server includes Retry-After in its response, use that value as the next polling interval.
    - This is critical for respecting server-side throttling.
- **Completion Status**

- When the process finishes, the status will change to complete.
- The response will include an **output array** containing URLs for bulk data files.

- **Availity Workflow**
  - Availity will download the files proactively.
  - Provide **proxy URLs** for each file to the new payer, ensuring secure and controlled access.





References and Resources

Environment URLs

The URLs for Availity's Payer-to-Payer Connectivity Hub environments are below. The non-production environment should be used for all testing purposes, as it includes the synthetic Old Payer and synthetic, PHI/PII-free patients. Once testing is complete and you are ready to go live, coordinate the switch to the Production environment with your Availity support representative.

Non-Production Environment: https://qua.api.availity.com

Production Environment: https://api.availity.com

Constraints within HL7 guides

Payer Identifiers

In the PDex implementation guide member-match request, the "coverage to match" is used to indicate who the Old Payer was. The identifier of the payer listed in CoverageToMatch is the value Availity expects for that target payer, used for routing API calls to the correct Old Payer. See Member-Match explanation for further details.

Minimum member information

- First Name
- Last Name
- Date of Birth
- Phone
- Address (postal code alone may be used as address)

Differentiation of New Payers

As the premise for this solution, an Old Payer will receive requests from multiple different New Payers through a single connection (Availity's Connectivity Hub). To differentiate requests coming from various payers, the Old Payer must look at the **CoverageToLink** section of a member match request for the New Payer's information.

Important Differences

Enterprises often use an API gateway to manage traffic with the outside world, so session authentication may look different for different payers. In addition, Availity's supported authentication

methods may change over time. Ensure that an agreed-upon authentication method is established during the implementation period.

The **X-P2P-Patient-Access-Token** header for patient-scoped token in **$everything** call is unique to Availity's implementation. See the $everything call explanation for details.

External References

The HL7 resources below may provide valuable background information as we work together to integrate your PDex solution with Availity's Payer-to-Payer Connectivity Hub.

HL7 Da Vinci PDex: https://build.fhir.org/ig/HL7/davinci-epdx/

HL7 Da Vinci HRex: https://hl7.org/fhir/us/davinci-hrex/STU1.1/

US Core 3.1.1: https://www.hl7.org/fhir/us/core/STU3.1.1/

US Core 6.1.0: https://hl7.org/fhir/us/core/STU6.1/

HL7 FHIR v4: http://hl7.org/fhir/R4/