**Prepared for:**

Centers for Medicare & Medicaid Services

and

Office of the National Coordinator for Health Information Technology

**CMS Alliance to Modernize Healthcare  
Federally Funded Research and Development Center**

Technical Authority for the Unified Clinical Quality   
Improvement Framework (Tacoma)

Bonnie Integration Application Programming Interface (API) Instructions

Draft

Version 1.0

August 7, 2018

The views, opinions, and/or findings contained in this report are those of The MITRE Corporation and should not be construed as official government position, policy, or decision unless so designated by other documentation.

This document was prepared for authorized distribution only. It has not been approved for public release.

© 2018, The MITRE Corporation. All Rights Reserved.

Record of Changes

| Version | Date | Author / Owner | Description of Change |
| --- | --- | --- | --- |
| 1.0 | August 7, 2018 | Chris Hossenlopp, Jason Walonoski, Peter Krautscheid, Lizzie Charbonneau | Initial documentation for the Bonnie Integration API v1 |

Table of Contents

[1. Background 4](#_Toc520981294)

[2. Integration Application Programming Interface 6](#_Toc520981295)

[2.1 API 7](#_Toc520981296)

[2.1.1 Measures 7](#_Toc520981297)

[2.1.2 Calculated Results 8](#_Toc520981298)

[2.2 Security 9](#_Toc520981299)

[2.2.1 Server-side (Bonnie) Configuration 10](#_Toc520981300)

[2.2.2 Client Integration 10](#_Toc520981301)

[2.2.3 Authorization Code Grant Type 10](#_Toc520981302)

[2.2.4 Security Tokens 11](#_Toc520981303)

[2.2.5 Token Information Endpoint 11](#_Toc520981304)

[3. Feedback and Support 12](#_Toc520981305)

[Appendix A. Code Samples 13](#_Toc520981307)

[A.1 Java Sample 13](#_Toc520981308)

[A.2 Ruby Sample 15](#_Toc520981309)

[Acronyms 16](#_Toc520981310)

List of Figures

[Figure 1. Manual Integration 2](#_Toc520918340)

[Figure 2. Bonnie Integration API 3](#_Toc520918341)

[Figure 3. Patient Results 6](#_Toc520918342)

List of Tables

[Table 1. Measure Operations 4](#_Toc520918358)

[Table 2. Measure Data Fields 5](#_Toc520918359)

[Table 3. Calculated Results Operations 6](#_Toc520918360)

# Background

The Centers for Medicare & Medicaid Services (CMS) and the Office of the National Coordinator for Health Information Technology (ONC) have collaborated to develop an alpha release of the Bonnie Integration Application Programming Interface (API). This API will enable applications to interact directly with Bonnie without using the user interface. The API is targeted for use by the Measure Authoring Tool (MAT), but could be used by any authorized application.

Currently, the Bonnie testing tool is only loosely integrated with the MAT through a manual export and import process. To test a measure in the MAT, the measure developer must first package and export the measure to their local file system via a measure bundle [in which the measure is expressed in a common digital format for encoding electronic clinical quality measures (eCQM)]. Once the file is exported, the developer then logs into the Bonnie application to load the bundle for testing. Figure 1 depicts the current integration process.

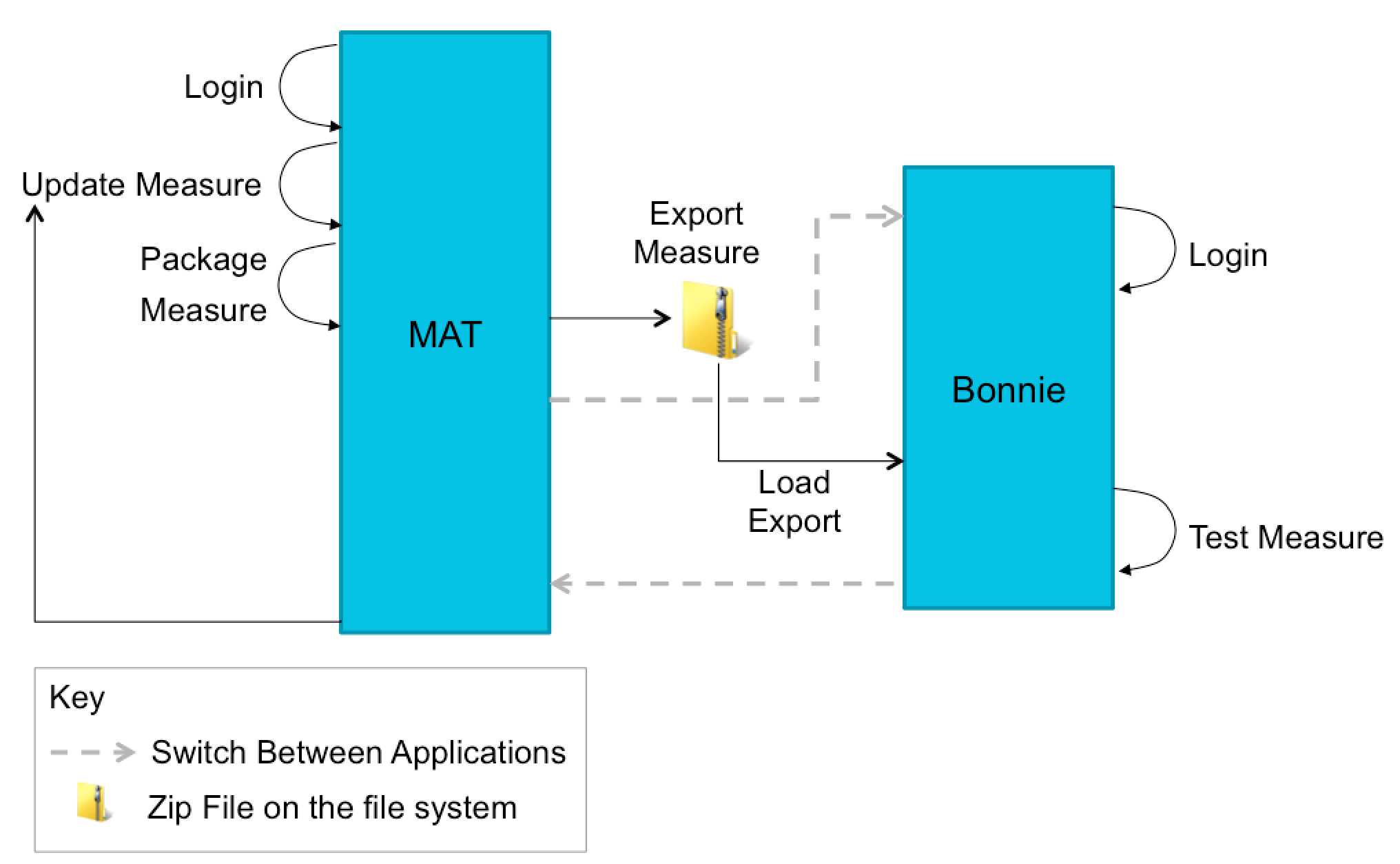


Figure 1. Manual Bonnie/MAT Integration

Manually exporting a measure from the MAT and importing it into Bonnie requires several steps, can be time consuming, and can fail because of user error. Given the need for iterative measure development, which can involve several cycles of updating a measure in the MAT and then testing the updated measure in Bonnie, the current process exponentially increases the time of measure development.

It is possible to simplify the iterative process of developing and then testing measures by improving the integration between the MAT and the Bonnie testing tool. By allowing the direct transmission of eCQMs between the tools, either by allowing Bonnie to pull eCQMs from the MAT (“pull integration”) or by allowing the MAT to push eCQMs into Bonnie (“push integration”), an integrated measure development process saves time, consequently lowering costs and increasing efficiency. In addition to these high-level integration approaches, other more minor integration improvements will further simplify the measure development process.

# Integration Application Programming Interface

The Bonnie team implemented an application programming interface (API) to enable push integration from the MAT (or other communicating system) to Bonnie. The Bonnie Integration API is a RESTful interface for authorized and authenticated clients to create, update, and read Measures, Patients, and calculated results. Once released, the primary documentation for the API will be available as an online reference: https://bonnie.healthit.gov/api.[[1]](#footnote-1)

This section addresses the Bonnie Integration API and Security. The API subsection describes the RESTful service, operations, end-points, and formats. The Security subsection describes the use and configuration of OAuth2, an open standard for authentication, to secure and access the service.

Figure 2 depicts the high-level system architecture from the perspective of the Bonnie infrastructure. A measure author interacts with both the MAT to create new measures and Bonnie to test those measures. Figure 4 shows two pathways—the Bonnie web access pathway on the reader’s left and the MAT pathway on the right. The MAT and other clients can access the same measures, patients, and calculated results available within Bonnie using OAuth2 security.

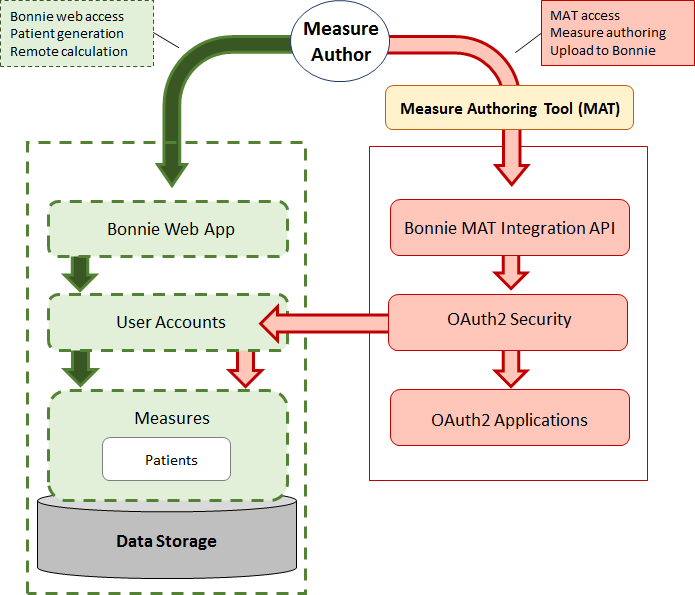


Figure 2. ****Bonnie Integration API****

## API

The Bonnie Integration API is RESTful: clients use HTTP verbs (e.g., GET, POST, PUT) on various endpoints (e.g., Measures) to create, update, and read resources. After a RESTful request is received, Bonnie will respond with the standard HTTP response status codes.[[2]](#footnote-2) If the response includes an error code, Bonnie will include an error message in the body of the response.

Measures are the top-level resource in the Bonnie Integration API structure. Clients may access a list of measures (filtered by their user credentials, as described in subsection on Security); create a new measure; update an existing measure; or drill down into a specific measure. Given a measure, a client can access the patient records associated with the measure or view the associated calculated results for each patient record.

This structure results in the following RESTful endpoints:

* {base}/api\_v1/measures
* {base}/api\_v1/measures/{id}
* {base}/api\_v1/measures/{id}/calculated\_results

For this list of endpoints, {base} should be substituted with the base Uniform Resource Locator (URL) of Bonnie, normally “https://bonnie.healthit.gov” (unless the client is using a private instance) and {id} should be substituted with an actual Health Quality Measures Format (HQMF) Set ID (for example, “40280381-3D61-56A7-013E-5CC8AA6D6290”).

The particular details associated with each endpoint follow in the Measures, Patient Records, and Calculated Results subsections, respectively.

### Measures

The measures endpoint allows clients to create, update, and read measures. Currently, there is no capability to delete measures outside of Bonnie.

For a create or update, the client must supply the complete measure (no partial creates or updates are supported) as multipart/form-data in the POST or PUT request.[[3]](#footnote-3) The multipart/form-data for the POST or PUT request should contain the MAT outputted measure package.

Table 1 lists the measure operations that are supported. Sample multipart/form-data examples are available on the official online reference documentation at: <http://bonn>ie/healthit.gov/api.[[4]](#footnote-4)

Table 1. Measure Operations

| Operation | HTTP Verb | Endpoint | Request Body |
| --- | --- | --- | --- |
| Create | POST | {base}/api\_v1/measures | multipart/form-data |
| Update | PUT | {base}/api\_v1/measures/{id} | multipart/form-data |
| Read List | GET | {base}/api\_v1/measures |  |
| Read Details | GET | {base}/api\_v1/measures/{id} |  |

Table 6 presents an outline description of the JSON data fields for measures that are returned after a GET request.

Table 2. Measure Data Fields

| Field | JSON Data Type | Format | Description |
| --- | --- | --- | --- |
| id | String | UUID | The RESTful id used to read the details of this measure, access the associated patients, or view the calculated results. |
| nqf\_id | String |  | An id assigned to this measure by the National Quality Forum. |
| hqmf\_id | String | UUID | The HQMF version id |
| hqmf\_set\_id | String | UUID | The HQMF id for this measure |
| hqmf\_version\_number | String |  | The HQMF version number |
| cms\_id | String |  | The id assigned to this measure by the Centers for Medicaid & Medicare Services. |
| title | String |  | The human-readable name. |
| description | String |  | A short or long human-readable description of this measure and logic. |
| type | String | “eh” or “ep” | eh – eligible hospital  ep – eligible provider |
| continuous\_variable | Boolean |  | Whether or not this is a continuous variable measure. |
| episode\_of\_care | Boolean |  | Whether or not this is an episode of care measure. |
| updated\_at | String | iso8601 | Date/Time stamp when this measure was last updated. |

### Calculated Results

The calculated results of a measure on the associated patient records are only accessible within the context of a measure. Calculated results are available as an Excel document, with each population or stratification corresponding to a sheet within the document. Table 3 depicts a sample calculated results operation.

Table 3. Calculated Results Operations

| Operation | HTTP Verb | Endpoint | Request Body |
| --- | --- | --- | --- |
| Read | GET | {base}/api\_v1/measures/{id}/calculated\_results |  |

The request header should have an Accept parameter as follows: “Accept: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet”.

Figure 3 shows a portion of an example Excel document containing patient calculation results.

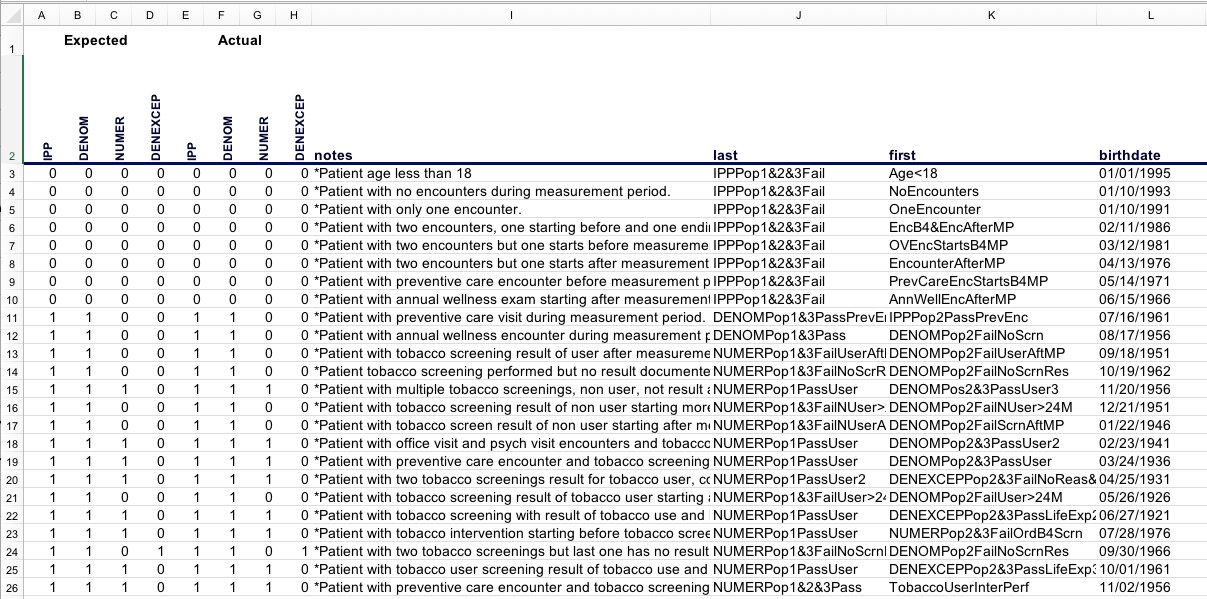


Figure 3. Patient Results

The Excel document will contain:

* expected and actual results for each population or stratification within the measure
* notes
* first and last names
* birthdate
* deathdate if applicable
* ethnicity, race, and gender
* calculation results for each definition statement within the CQL for each population or stratification within the measure.

## Security

The Bonnie Integration API is secured using OAuth2[[5]](#footnote-5). To assure client access to measure and patient data, the client application must be pre-configured on the Bonnie server. When the user uses the client application to access Bonnie, they must present valid Bonnie user credentials. The server-side and client-side configuration and workflows are explained separately.

For an application to gain access to Bonnie through the Bonnie Integration API, they should email the Bonnie development team at [bonnie-feedback-list@mitre.org](mailto:bonnie-feedback-list@mitre.org) to start the process and receive the appropriate credentials.

### Server-side (Bonnie) Configuration

To authorize a client application to access the Bonnie Integration API, a Bonnie Administrator needs to create a record of the client application using the following steps:

1. Administrator logs into Bonnie.
2. Administrator navigates to <https://bonnie.healthit.gov/oauth/applications>[[6]](#footnote-6).
3. Administrator clicks the “New Application” button.
4. Administrator fills out the new application form:

* **Name:** enter the application name, e.g., “MAT”
* **Redirect Uniform Resource Identifier (URI):** enter “https://bonnie.healthit.gov/api\_v1/measures” if using Resource Owner Password Credentials flow. Otherwise, when using Authorization Code flow, enter the application URI where Bonnie will redirect the web browser to finish authorization.
* **Confidential:** check the checkbox.
* **Scopes:** leave blank for default scopes.

1. Administrator clicks the “Submit” button.
2. The Application is created, providing an “Application Id” and “Secret”. These alphanumeric fields should be securely provided to the client application development team.

### Client Integration

If a client application has been granted an Application ID and Secret, and valid Bonnie user credentials are available, a client can access data for that one user.

The OAuth2 specification defines four authorization grant types:

1. Authorization Code
2. Implicit
3. Resource Owner Password Credentials
4. Client Credentials

The Bonnie Integration API requires authorization grant type #1, Authorization Code, as measures and patients are scoped within a user account.

### Authorization Code Grant Type

The authorization code grant type[[7]](#footnote-7) allows for the client application of the Bonnie Integration API to avoid direct access to the user’s Bonnie username and password. This grant type flow is what is commonly considered the OAuth flow. The following overview demonstrates how the flow works:

1. The client application begins by directing the user’s browser to the Bonnie API OAuth authorization endpoint.

* <https://bonnie.healthit.gov/oauth/authorize>? response\_type=code&client\_id=CLIENT\_ID&redirect\_uri=REDIRECT\_URI*[[8]](#footnote-8)*
* The *CLIENT\_ID* is the Application Id provided by the Bonnie OAuth admin (see Step 6 of the Server-side (Bonnie) Configuration subsection).
* The *REDIRECT\_URI* must match the redirect URI configured for the application (see Step 4 of the Server-side (Bonnie) Configuration subsection).

1. The user will authenticate with Bonnie if they are not already authenticated. The user will be asked to approve the application for access if this is their first time connecting the application to the Bonnie Integration API.
2. Bonnie will redirect the user’s browser to the client application’s authorization *REDIRECT\_URI* with the authorization code provided in the URL parameters.
3. The client application will send a request to the Bonnie OAuth token endpoint (<https://bonnie.healthit.gov/oauth/token)> providing the authorization code received in the request URL parameters along with the *REDIRECT\_URI* and *CLIENT\_ID*. The token endpoint returns the OAuth Access Token and the OAuth Refresh Token. See section 2.2.4 for further explanation.

### Security Tokens

There are two types of tokens that Bonnie provides to the client application: OAuth Access Tokens and OAuth Refresh Tokens. The OAuth Access Tokens are used for requests to the API. The OAuth Access Tokens will expire after some set amount of time, currently configured for 15 minutes. The OAuth Refresh Tokens are issued at the same time as the OAuth Access Tokens. The OAuth Refresh Tokens are used to retrieve a new OAuth Access Token. The OAuth Refresh Tokens are currently configured to last for five days, and are revoked after first use.

The Bonnie token revocation process follows the OAuth 2.0 revocation specification[[9]](#footnote-9). A user of a client application can also revoke access of the client application to their Bonnie account by navigating to <https://bonnie.healthit.gov/oauth/authorized_applications>.

### Token Information Endpoint

Bonnie has implemented a Token Information Endpoint, accessible at <https://bonnie.healthit.gov/oauth/token/info>. This endpoint provides information about the resource owner (the user of the Bonnie account that the token is associated with) including the first name, last name, and email; when the token was created; the length of time in seconds until the OAuth Access Token expires; and the length of time in seconds until the OAuth Refresh Token expires.

# Feedback and Support

An issue tracker and feedback email list are available to support the resolution of issues and to answer questions related to the Bonnie Integration API. The Bonnie issue tracker is available on the ONC Jira system at: <https://oncprojectracking.healthit.gov/support/projects/BONNIE>.

Users can also reach out to the Bonnie team through the Bonnie feedback list at [bonnie-feedback-list@mitre.org](mailto:bonnie-feedback-list@mitre.org). The Bonnie feedback list email can be accessed using the “Contact” link in the main Bonnie navigation menu at the top of every page.

1. Code Samples
   1. Java Sample

The Java Sample uses the Apache Oltu library to execute the Resource Owner Password Credentials grant flow. For illustration purposes, in the following Listing 1 example of a Java client, the OAuthAccessTokenResponse class is extended to override the JSON response parsing:

**package org.mitre.bonnie;**

**import** org.apache.oltu.oauth2.client.OAuthClient;

**import** org.apache.oltu.oauth2.client.URLConnectionClient;

**import** org.apache.oltu.oauth2.client.request.OAuthBearerClientRequest;

**import** org.apache.oltu.oauth2.client.request.OAuthClientRequest;

**import** org.apache.oltu.oauth2.client.response.OAuthResourceResponse;

**import** org.apache.oltu.oauth2.common.OAuth;

**import** org.apache.oltu.oauth2.common.exception.OAuthProblemException;

**import** org.apache.oltu.oauth2.common.exception.OAuthSystemException;

**import** org.apache.oltu.oauth2.common.message.types.GrantType;

**public** **class** Client {

**private** **static** **final** String **client\_id** = "\*\*\*\*\*\*";

**private** **static** **final** String **secret** = "\*\*\*\*\*\*

**private** **static** **final** String **email** = "bonnie@example.com";

**private** **static** **final** String **password** = "\*\*\*\*\*";

**private** **static** **final** String **server\_base** = "https://bonnie.healthit.gov";

**private** **static** **final** String **api\_url** = "/api\_v1/measures";

**private** **static** **final** String **token\_url** = "/oauth/token";

**public** **static** **void** main(String[] args)

{

**try** {

// Create an OAuth2 request for a token

OAuthClientRequest request = OAuthClientRequest

.tokenLocation(Client.**server\_base** + Client.**token\_url**)

.setGrantType(GrantType.**PASSWORD**)

.setClientId(Client.**client\_id**)

.setClientSecret(Client.**secret**)

.setUsername(Client.**email**)

.setPassword(Client.**password**)

.setRedirectURI(Client.**server\_base** + Client.**api\_url**)

.buildBodyMessage();

// Create an OAuth2 client

OAuthClient client = **new** OAuthClient(**new** URLConnectionClient());

// And send the token request...

BonnieAccessTokenResponse tokenResponse = client.accessToken(request, BonnieAccessTokenResponse.**class**);

// With the token response, we can now make requests against the MAT API...

OAuthClientRequest measuresRequest = **new** OAuthBearerClientRequest(Client.**server\_base** + Client.**api\_url**)

.setAccessToken(tokenResponse.getAccessToken()).buildQueryMessage();

// Use the client to send the MAT API request...

OAuthResourceResponse resourceResponse = client.resource(measuresRequest, OAuth.HttpMethod.**GET**, OAuthResourceResponse.**class**);

// Dump the response to the console...

System.**out**.println(resourceResponse.getBody());

} **catch** (OAuthSystemException e) {

e.printStackTrace();

} **catch** (OAuthProblemException e) {

e.printStackTrace();

}

}

}

The following Listing 2 example demonstrates the Java Access Token Parser:

**package** org.mitre.bonnie;

**import** java.io.StringReader;

**import** javax.json.Json;

**import** javax.json.JsonObject;

**import** javax.json.JsonReader;

**import** org.apache.oltu.oauth2.client.response.OAuthAccessTokenResponse;

**import** org.apache.oltu.oauth2.common.exception.OAuthProblemException;

**import** org.apache.oltu.oauth2.common.token.BasicOAuthToken;

**import** org.apache.oltu.oauth2.common.token.OAuthToken;

**public** **class** BonnieAccessTokenResponse **extends** OAuthAccessTokenResponse {

**public** String body = **null**;

**public** String contentType = **null**;

**public** **int** responseCode = 0;

**private** String accessToken = **null**;

**private** **long** expires\_in = 0;

**private** OAuthToken token = **null**;

**private** String refreshToken = **null**;

**private** String scope = **null**;

@Override

**public** String getAccessToken() { **return** **this**.accessToken; }

@Override

**public** Long getExpiresIn() { **return** **this**.expires\_in; }

@Override

**public** OAuthToken getOAuthToken() { **return** **this**.token; }

@Override

**public** String getRefreshToken() { **return** **this**.refreshToken; }

@Override

**public** String getScope() { **return** **this**.scope; }

@Override

**protected** **void** setBody(String arg0) **throws** OAuthProblemException { **this**.body = arg0; }

@Override

**protected** **void** setContentType(String arg0) { **this**.contentType = arg0; }

@Override

**protected** **void** setResponseCode(**int** arg0) { **this**.responseCode = arg0; }

**protected** **void** validate(){

**if**(body!=**null**) {

JsonReader reader = Json.*createReader*(**new** StringReader(**this**.body));

JsonObject object = (JsonObject) reader.read();

**this**.accessToken = object.getString("access\_token");

**this**.expires\_in = object.getJsonNumber("expires\_in").longValue();

**this**.token = **new** BasicOAuthToken(**this**.accessToken,**this**.expires\_in);

}

}

}

* 1. Ruby Sample

The following Ruby sample demonstrates the Resource Owner Password Credentials grant flow and requires the “oauth2” gem:

require 'oauth2'

client\_id = "CLIENT\_ID\_GOES\_HERE"

secret = "CLIENT\_SECRET\_GOES\_HERE"

username = "bonnie@example.com"

password = "USER\_PASSWORD\_GOES\_HERE"

options = {

:site => "https://bonnie.healthit.gov",

:authorize\_url => "https://bonnie.healthit.gov/oauth/authorize",

:token\_url => "https://bonnie.healthit.gov/oauth/token",

:raise\_errors => true

}

client = OAuth2::Client.new(client\_id,secret,options)

token = client.password.get\_token(username,password)

response = token.get("https://bonnie.healthit.gov/api\_v1/measures")

puts response.status

puts response.body

Acronyms

| Acronym | **Definition** |
| --- | --- |
| API | Application Programming Interface |
| eCQM | Electronic Clinical Quality Measure |
| CMS | Centers for Medicare & Medicaid Services |
| FISMA | Federal Information Security Management Act |
| HCIS | Health Care Innovation Services |
| HQMF | Health Quality Measures Format |
| MAT | Measure Authoring Tool |
| OAuth | Open standard to authorization |
| ONC | Office of the National Coordinator for Health Information Technology |
| QDM | Quality Data Model |
| REST | Representational State Transfer |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| XML | Extensible Markup Language |

1. This link will be active when the API is released to the Production server. A release date has not yet been scheduled as of the latest version of this document. [↑](#footnote-ref-1)
2. For more information on HTTP response codes, please see the Request for Comments (RFC) for HTTP, section 6: <https://tools.ietf.org/html/rfc7231#section-6>. [↑](#footnote-ref-2)
3. For more information on multipart/form-data, please see the RFC for multipart/form-data located here: <https://www.ietf.org/rfc/rfc2388.txt>. [↑](#footnote-ref-3)
4. This link will be active when the API is released to the Production server. A release date has not yet been scheduled as of the latest version of this document. [↑](#footnote-ref-4)
5. [RFC6749](https://tools.ietf.org/html/rfc6749) provides the details of the OAuth2 specification [↑](#footnote-ref-5)
6. This link will be active when the API is released to the Production server. A release date has not yet been scheduled as of the latest version of this document. [↑](#footnote-ref-6)
7. [RFC6749§4.1](https://tools.ietf.org/html/rfc6749#section-4.1) provides details. [↑](#footnote-ref-7)
8. The base URL and path will be active when the API is released to the Production server. A release date has not yet been scheduled as of the latest version of this document. [↑](#footnote-ref-8)
9. [RFC7009](https://tools.ietf.org/html/rfc7009) provides details. [↑](#footnote-ref-9)