Centers for Medicare & Medicaid Services

Office of the National Coordinator for
Health Information Technology

Bonnie Integration Application Programming Interface (API) **Alpha Release Instructions**

Version 1.0

July 8, 2016

Table of Contents

List of Figures

List of Tables

1. 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). At present, the Bonnie testing tool is only loosely integrated with the Measure Authoring Tool (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.

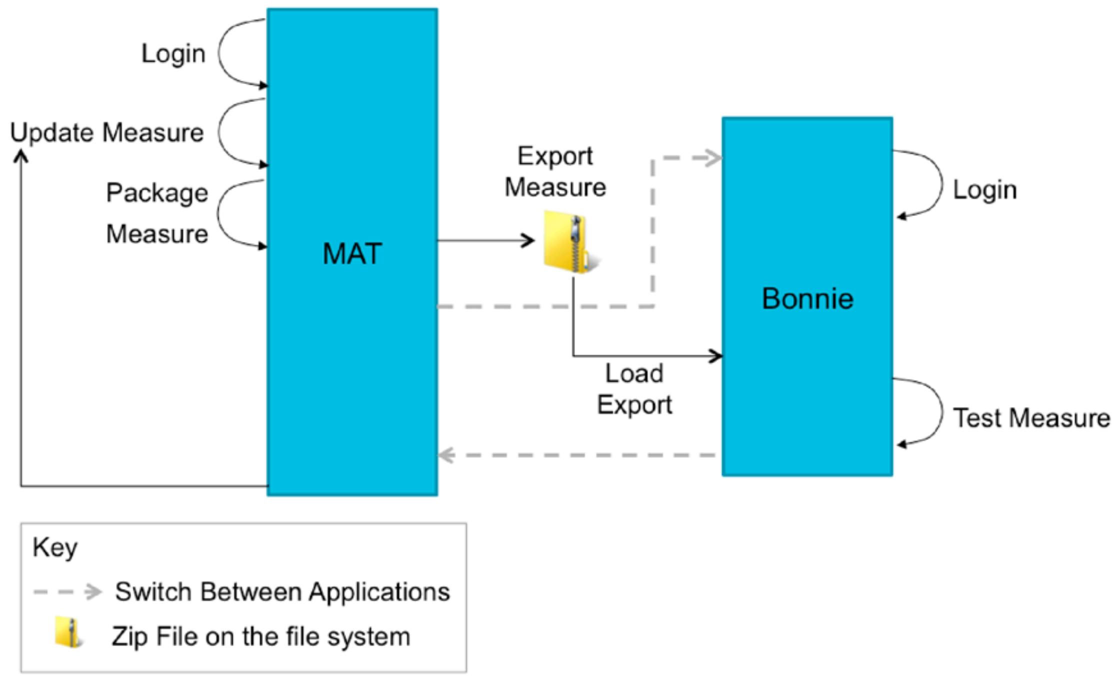

Figure 1. Manual Integration

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.

This document describes the **alpha** release of the Bonnie Integration API, enabling “push integration” from the MAT to Bonnie. The Bonnie Integration API is **subject to change before a formal release is made**.

2. Integration Application Programming Interface

Integrating the Bonnie and MAT applications requires a programmatic interface allowing the two applications to communicate. The Bonnie MAT Integration API is a Representational State Transfer (REST) 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.ahrqstg.org/api/1

This section addresses the MAT 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.

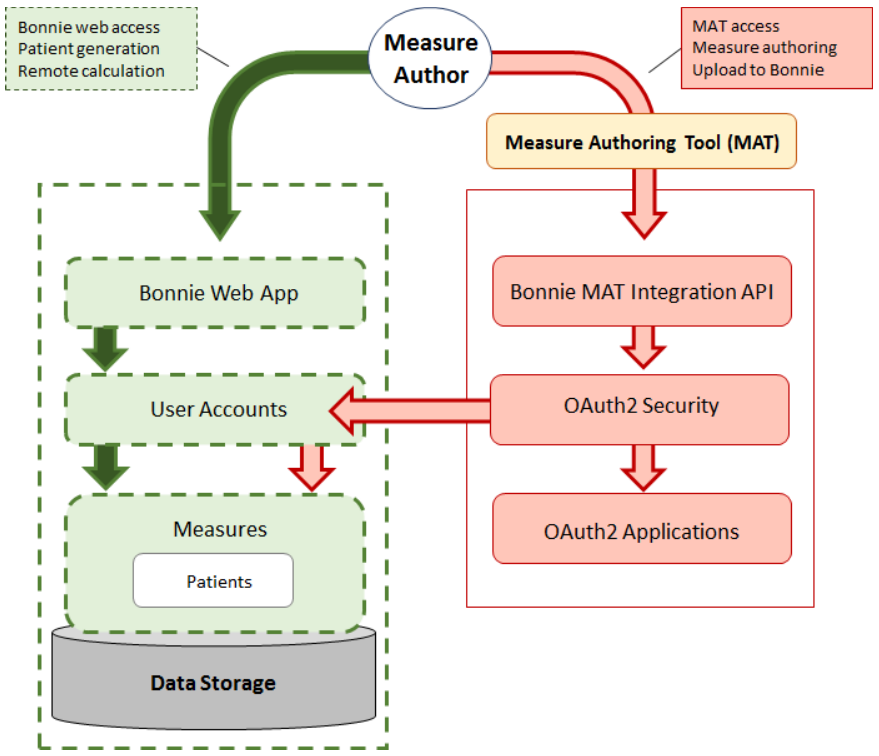

Figure 2. MAT Integration API

2.1 API

The MAT 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.

Measures are the top-level resource in the MAT Integration API structure. Clients can access a list of measures (filtered by their user credentials; see 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}/patients
- {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.ahrqstg.org" (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.

2.1.1 Measures

The measures endpoint allows clients to create, update, and read measures. Currently, there is no ability 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. Table 1 lists the measure operations that are supported. Sample multipart/form-data examples are available on the official online reference documentation at: http://bonnie/healthit.gov/api/1.

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 2 presents an outline description of data fields for measures.

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 Foundation.
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.
name	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
category	String		A human-readable category.
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.
complexity	Object		
complexity.populations	Array of Objects		List of populations associated with this measure.
complexity.populations.name	String		Human-readable population name.
complexity.populations.complexity	Number		Complexity value of the population.
complexity.variables	Array		List of variables associated with this measure.
updated_at	String	iso8601	Date/Time stamp when this measure was last updated.

2.1.2 Patient Records

Patient records are only accessible within the context of a measure. Patient records are available as a list as shown in Table 3, but are not available on an individual basis.

Table 3. Patient Operations

Operation	HTTP Verb	Endpoint	Request Body
Read List	GET	{base}/api_v1/measures/{id}/patients	

Table 4 presents an outline description of data fields for patient records.

Table 4. Patient Data Fields

Field	JSON Data Type	Format	Description
_id	String	Base64	The id used to identify a patient.
birthdate	String	iso8601	Date/Time stamp the patient was born.
description	String		The human-readable description of the patient.
ethnicity	Object		Patient ethnicity (as a “codeable concept”)
ethnicity.code	String		The ethnicity code.
ethnicity.name	String		The ethnicity name.
ethnicity.codeSystem	String		The coding system that the ethnicity code is from.
expected_values	Array of Objects		List of expected values by measure population.
expected_values.population_index	Number		The index into the list of measure populations.
expected_values.IPP	Number		Initial Patient Population *****
expected_values.DENOM	Number		Denominator *****
expected_values.NUMER	Number		Numerator *****
expected_values.DENEX	Number		Denominator Exclusion *****
expected_values.DENEXCEP	Number		Denominator Exception *****
expired	Boolean		Whether the patient is expired, true or false.
first	String		First or given name.
last	String		Last or surname.
gender	String	M,F	Patient gender.
insurance_providers	Array of Objects		Patient insurance providers.
insurance_providers.member_id	String		Insurance membership id.

Field	JSON Data Type	Format	Description
insurance_providers.payer	String		Insurance Payer detail.
race	Object		Patient race (as a “codeable concept”)
race.code	String		The race code.
race.name	String		The race name.
race.codeSystem	String		The coding system that the race code is from.
updated_at	String	iso8601	Date/Time stamp when this patient was last updated.

* **Value greater than zero indicates that the patient is expected to be in the population group.**

2.1.3 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 a list, with each item corresponding to a patient record. Table 5 depicts a sample calculated results operation.

Table 5. Calculated Results Operations

Operation	HTTP Verb	Endpoint	Request Body
Read	GET	{base}/api_v1/measures/{id}/calculated_results	

Table 6 outlines a description of data fields for calculated results.

Table 6. Calculated Results Data Fields

Field	JSON Data Type	Format	Description
status	String	“complete” or “error”	The status of the calculation. Error indicates problems with the measure.
messages	Array of String		List of messages if the status equals “error”
measure_id	String	UUID	The measure id.
summary	List		List of population summaries.
summary.IPP	Number		Total patients within the Initial Patient Population
summary.DENOM	Number		Total patients within the Denominator
summary.NUMER	Number		Total patients within the Numerator
summary.DENEX	Number		Total patients within the Denominator Exclusion

Field	JSON Data Type	Format	Description
summary.DENEXCEP	Number		Total patients within the Denominator Exception
patient_count	Number		The total number of patients used in the calculation.
patients	Array		List of patients, each Patient contains the data fields listed in Table 9, and the "actual_values" that follow**.**
patients.actual_values	Array of Objects		List of actual values by measure population.
patients.actual_values.IPP	Number		Initial Patient Population *****
patients.actual_values.DENOM	Number		Denominator *****
patients.actual_values.NUMER	Number		Numerator *****
patients.actual_values.DENEX	Number		Denominator Exclusion *****
patients.actual_values.DENEXCEP	Number		Denominator Exception *****

* **Value greater than zero indicates that the patient was counted in the population group.**

2.2 Security

The MAT Integration API is secured using OAuth2. To assure client access to measure and patient data, the client application must be pre-configured on the Bonnie server and must present valid Bonnie user credentials. The server-side and client-side configuration and workflows are explained separately.

2.2.1 Server-side (Bonnie) Configuration

To authorize a client application to access the MAT 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.ahrqstg.org/oauth/applications
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.ahrqstg.org/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.
    - **Scopes**: leave blank for default scopes.

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

2.2.2 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 MAT Integration API requires #1 or #3—Authorization Code or Resource Owner Password Credentials—as measures and patients are scoped within a User account.

2.2.3 Authorization Code Grant Type

The authorization code grant type allows for the client application of the MAT 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. See RFC6749§4.1 for details.

1. The client application begins by directing the user’s browser to the Bonnie API OAuth authorization endpoint.
    - https://bonnie.ahrqstg.org/oauth/authorize?response_type=code&client_id=*CLIENT_ID*&redirect_uri=*REDIRECT_URI*
    - 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).
2. 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 MAT Integration API.
3. Bonnie will redirect the user’s browser to the client application’s authorization *REDIRECT_URI* with the authorization code provided in the URL parameters.
4. The client application will send a request to the Bonnie OAuth token endpoint (https://bonnie.ahrqstg.org/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 that can be used for further requests to the API.

2.2.4 Resource Owner Password Credentials

The Resource Owner Password Credentials grant type is a simple flow that allows the client application to obtain an Access Token using one request. With this flow the client application has access to the user's Bonnie username and password. See RFC6749§4.3 for details (available at: https://tools.ietf.org/html/rfc6749#section-4.3). The following samples show how to use this grant type.

2.2.5 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.ahrqstg.org";
  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);
             }
       }
}
```

2.2.6 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.ahrqstg.org",
  :authorize_url => "https://bonnie.ahrqstg.org/oauth/authorize",
  :token_url => "https://bonnie.ahrqstg.org/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.ahrqstg.org/api_v1/measures")
puts response.status
puts response.body
```

Acronyms

API	Application Programming Interface
eCQM	Electronic Clinical Quality Measure
CMS	Centers for Medicare & Medicaid Services
HQMF	Health Quality Measures Format
MAT	Measure Authoring Tool
OAuth	Open standard for authentication
ONC	Office of the National Coordinator for Health Information Technology
REST	Representational State Transfer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator