PharmaSUG 2024 - Paper MM-245

Relax with Pinnacle 21's RESTful API

Trevor Mankus, Certara

ABSTRACT

Harness the power of Pinnacle 21 Enterprise's REST API and relax! Programmatically extract metadata from within your validation environment, from study level specifications to broader organizational-level metadata like as Standards and Terminologies. The metadata retrieved includes datasets, variables, value level, codelists, terms, methods, comments, and even Analysis Results Metadata (ARM) for ADaM studies. These objects can then be incorporated into the programming process, such as setting up attributes for variables, assigning dataset labels, establishing key variables, and pairing coded and decoded values. This paper will detail how to access the Pinnacle 21 Enterprise REST API and discuss best practices on implementation within the data standardization process. It will also cover the various options for export file formats as well as provide examples for practical use of metadata to ensure automating metadata retrieval is a breeze.

INTRODUCTION

In a few lines of code, the computer program of your choice can request and receive metadata directly from your P21E environment. The results can be returned in various digestible formats that can be easily parsed through to aide in the development of your programming code. This metadata can then be used to build tools for automation, drive dataset and variable attributes, and even help assign controlled terminology within your program. Automated programs can be used to request metadata not only for individual Data Packages (Defines), but for Standards and Terminologies, as well.

WHAT IS AN API

REST API stands for **Re**presentational **S**tate **T**ransfer **A**pplication **P**rogramming **I**nterface. Our REST API works essentially the same way that any website does. A call is made from a client to the P21E server, and data is received back over the HTTP/HTTPS protocol. Each URL call is known as a request while the data sent back to you is called a response. An API request to a REST API is comprised of the following components:

- **Request headers**: API request headers are key-value pairs that provide additional information about the request. For instance, the Authorization header provides authentication credentials, such as an API Key or OAuth token, to authenticate the requester.
- **Endpoint**: Every request points to an API endpoint, which is a URL that provides access to a resource. For example, the `/products` endpoint in an e-commerce app may include the logic for processing all requests that are related to products. The reason an endpoint is required is because without it, the API will not know how to proceed (i.e., what to do with the request).
- **Method**: Every request must include a method which defines the operation that the client would like to perform on the specified resource. REST APIs are accessible through standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE, which facilitate common actions like retrieving, creating, updating, or deleting data. Currently, the P21E API only supports GET methods.
- **Parameters**: Parameters are the variables that are passed to an API endpoint to provide specific instructions for the API to process. These parameters can be included in the API request as part of the URL. For example, the `/products` endpoint of an e-commerce API might accept a "color" parameter, which it would use to access and return products of a specific color.

PINNACLE 21 ENTERPRISE - API ENDPOINTS

P21E currently supports the retrieval of Study Specs (Defines), Standards, and Terminologies. In addition, requests to return a list of all objects of a given type are allowed as well. You'll form the call for the content you need simply by putting together the following content as seen in Table 1.

Metadata Type	API Endpoint	Notes
Study Specs (Defines)	`/api/defines`	Retrieve a list of all Study Specs that API Key has access to
	`/api/defines/:id/export`	Retrieve content of Study Spec for data package <id>
Standards	`/api/standards`	Retrieve a list of all Standards in the P21E environment
	`/api/standards/:id/export`	Retrieve content of Standard <id>, including all dataset, variable, and value level metadata
Terminologies	`/api/terminologies`	Retrieve a list of all Terminologies in the P21E environment
	`/api/terminologies/:id/export`	Retrieve content of Terminology <id>, including codelists and terms

Table 1. API endpoints for various metadata types

The API endpoint to get the list of objects returns the response in JSON format. These results include details for each specific object including name, version, description, type, publisher, and the ID itself. You can use these results to scan through and find the content you are interested in, and then build a second API call using the <id> to pull down the content. For example, Output 1 shows what `/api/standards` will return in the JSON result:


```
{
  "message":"success",
  "datetime":"2024-02-21 09:33:33",
  "list":[
   {
     "exportHref":"/api/standards/5295/export",
     "name":"SDTMIG",
     "version":"3.1.2",
     "description":"Study Data Tabulation Model Implementation Guide",
     "type":"SDTM",
     "publisher":"CDISC",
     "modelStandard":"SDTM",
     "modelStandardVersion":"1.2",
     "status":"Final",
     "lastPublished":"2023-03-02 09:33:33",
     "lastUpdated":"2023-03-02 09:33:33"
   },
   ...
  ]
}
```


Output 1. Example of the JSON formatted results for /api/standards

You can parse through this JSON output to find the Standard you are interested in, and then use the value found in the `exportHref` key to grab the complete content. For example, if you were trying to export the metadata from SDTMIG 3.1.2, you could send a second API request to

`/api/standards/5295/export` now that you've successfully found the appropriate ID value. Note that this ID also matches the ID found in the URL when viewing the Standard in your web browser.

PINNACLE 21 ENTERPRISE - API ENDPOINT PARAMETERS

The P21E API has two options you can specify to customize the response (see Table 2). These are passed in as query parameters in the request itself.

Parameter	Description	Valid Options
`format`	File format of the requested metadata	json (default), xml, pdf, xlsx
`option`	If provided, the exported Excel workbook will be one dataset per worksheet. Only valid in combination with format = xlsx.	byDataset

Table 2. Query parameters in the P21E REST API

Note that if no query parameters are specified, `format=json` is used as the default. When `format=xml`, the response is a define.xml formatted file—the same one you would get if you went through the P21E user interface and exported a define.xml. When `format=pdf`, the response is a define.pdf formatted file—the same one you would get if you went through the P21E user interface and exported a define.pdf.

To include query parameters, add a question mark (`?`) after the request and specify the parameter and value. If more than one parameter is specified, separate them with an ampersand (`&`). For example:

```
/api/standards/5295/export?format=xlsx
```

The request above will export the Standard metadata for #5295 in Excel format using our P21E excel template where all variable level metadata is in a worksheet named Variables. Alternatively, you can add a second option to the request to split the Excel workbook into dataset-named worksheets.

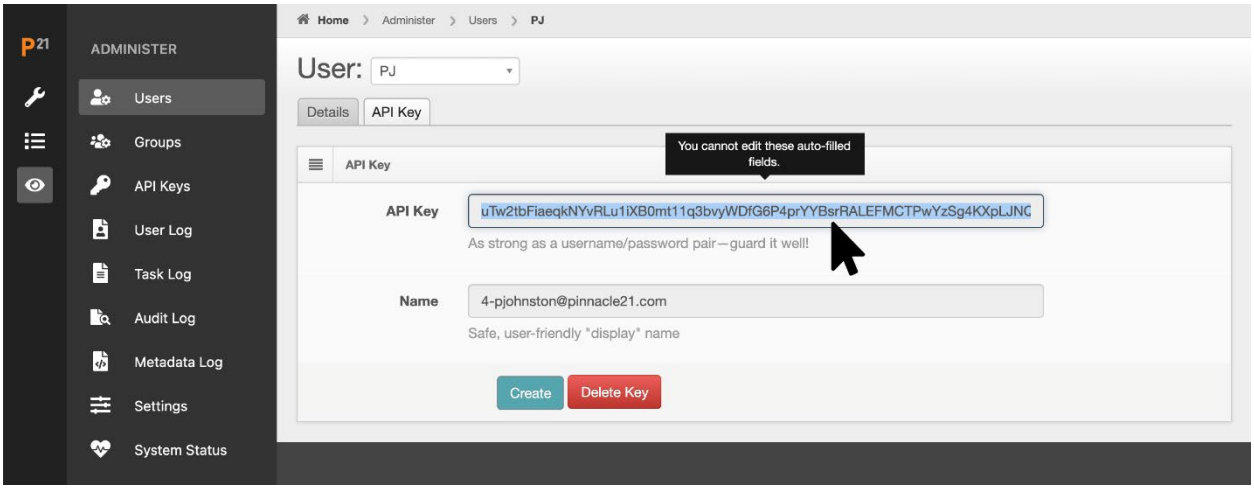
```
/api/standards/5295/export?format=xlsx&options=byDataset
```

API KEYS AND P21 ENTERPRISE

Pinnacle 21's REST API utilizes the same authentication technology that the Enterprise Command Line Interface (ECLI) uses – **API Keys**. Currently, P21E supports both system-wide as well as user-specific API Keys. System API Keys are unrelated to any human or user account. The benefit of a System API Key is that it's just one set of credentials to manage. However, sharing a common, anonymous System API Key is not ideal. Unlike a System API Key, a User API Key mirrors its named User perfectly. Whatever is true of the User is always also true of their User API Key. This results in multiple benefits:

1. The full string of the Key is only ever shown to that User and appears as [hidden] even to other Users with the Administrator Role—since it is as strong as a username/password pair (see Display 1).
2. The User API Key 100% matches the Roles/Groups/Assignments of that User.
3. In the Audit Log and Validation Log, actions performed by a User API Key are identifiable by their named User. E.g., User = PJ.

Deciding between a System API Key and a User API Key is entirely based on your usage requirements. If you decide to utilize User API Keys, each user will need to provide their unique credentials during the API call itself. If you decide to use a System API Key, a single set of authentication credentials can be used.



Display 1. Example of a User API Key for "PJ"

An API Key is as strong as a username/password pair—guard it well! It grants access to your P21E environment.

ENCODING THE API KEY FOR AUTHENTICATION

Basic authentication is typically used in conjunction with HTTPS to provide confidentiality and requires your credentials (i.e., P21E API Key) to be the Base64 encoding of ID and Password. The format for passing in your username and password is <username>:<password>. Since we are using an API Key, both are the API Key itself.

You can encode your API Key in SAS or R using built in functions. For example, in SAS you can use the code below to store the encoded API key in a macro variable `&apikey`:

```
* Encode the API Key in Base64 as user:pass;
data _null_;
 call symput("apikey", "<API Key>:<API Key>", $base64x500.));
run;
```

PROC HTTP IN SAS

The HTTP procedure (PROC HTTP) allows you to invoke a web service that issues requests.

```
PROC HTTP URL="URL-to-target</redirect/n>"
    <METHOD=<">http-method<">>
    <authentication-type-options>
    <caching-options>
    <header-options>
    <proxy-server-connection-options>
    <web-server-authentication-options>
    <EXPECT_100_CONTINUE>
    <FOLLOWLOC | NOFOLLOWLOC>
    <HTTP_TOKENAUTH>
    <IN=<fileref | FORM (arguments) | MULTI <options> (parts) | "string">>
    <MAXREDIRECTS=n>
    <OUT=fileref>
    <QUERY=("parm1"="value1" "parm2="value2" ...)>
;
```

We can simplify this down to only include the required arguments for our needs. At a minimum, you need to specify the URL, METHOD, output destination, and authentication credentials.

```
filename resp temp;
proc http
 url  = "https://p21e.pinnacle21.com/api/standards/5295/export"
 method = "GET"
 out  = resp;
 headers
   "Authorization" = "Basic %trim(&apikey.)";
run;
```

In the example above, the endpoint is defined in the URL argument, the method (GET) is specified in the method argument, output is pointing to a temporary filename named 'resp,' and we pass in a single request header named 'Authentication' using the Basic scheme. The value passed in for our API Key is the base64 encoded value of `user:pass` (<API Key>:<API Key>).

SAS also provides a means to check that the request was made successfully. You can use the system-generated macro variable `&SYS_PROCHTTP_STATUS_CODE` to verify the response is 200 (OK). Any other status code would indicate a problem with the request. HTTP status codes let you tell the status of the response quickly. They range from 100+ to 500+, and in general, the numbers follow these rules:

- 200 means the request has succeeded
- 300+ means the request is redirected to another URL
- 400+ means an error that originates from the client has occurred
- 500+ means an error that originates from the server has occurred

Common HTTP status codes include 400 (bad request) and 403 (forbidden/access denied). In these cases, check that you supplied the correct endpoint and query parameters and ensure the API Key specified has the appropriate access to the content being requested.

JSON FILE FORMAT AND SAS

SAS can process JSON results very easily using a single libname statement.

```
libname api JSON fileref=resp;
```

This line will set up a new library named 'api,' which can parse through the results of the JSON content found in temporary fileref 'resp'. Once configured, you can then easily copy all the content from the 'api' libname over into your WORK directory.

```
proc copy in=api out=work memtype=data;
run;
```

If run successfully, you should see the following in your LOG file:

```
NOTE: PROCEDURE HTTP used (Total process time):
   real time      0.58 seconds
   cpu time      0.01 seconds

NOTE: 200 OK

NOTE: Libref API was successfully assigned as follows:
   Engine:    JSON
   Physical Name: C:\Users\tmankus\AppData\Local\Temp\SAS Temporary
Files\_TD3320_CER-PF46VBDT_\#LN00011

NOTE: Copying API.ALLDATA to WORK.ALLDATA (memtype=DATA).
NOTE: Copying API.ANALYSIS_DISPLAYS to WORK.ANALYSIS_DISPLAYS (memtype=DATA).
NOTE: Copying API.ANALYSIS_RESULTS to WORK.ANALYSIS_RESULTS (memtype=DATA).
NOTE: Copying API.CODELISTS to WORK.CODELISTS (memtype=DATA).
NOTE: Copying API.COMMENTS to WORK.COMMENTS (memtype=DATA).
NOTE: Copying API.DATASETS to WORK.DATASETS (memtype=DATA).
NOTE: Copying API.DEFINE to WORK.DEFINE (memtype=DATA).
```

```
NOTE: Copying API.DICTIONARIES to WORK.DICTIONARIES (memtype=DATA).
NOTE: Copying API.DOCUMENTS to WORK.DOCUMENTS (memtype=DATA).
NOTE: Copying API.METHODS to WORK.METHODS (memtype=DATA).
NOTE: Copying API.ROOT to WORK.ROOT (memtype=DATA).
NOTE: Copying API.TERMS to WORK.TERMS (memtype=DATA).
NOTE: Copying API.VALUE_LEVEL to WORK.VALUE_LEVEL (memtype=DATA).
NOTE: Copying API.VARIABLES to WORK.VARIABLES (memtype=DATA).
NOTE: PROCEDURE COPY used (Total process time):
   real time      0.09 seconds
   cpu time       0.07 seconds
```

Output 2. Content from SAS LOG after PROC HTTP, LIBNAME, and PROC COPY

These datasets are now viewable in your SAS instance.

- **ALLDATA**: Contains all the content from the JSON response, stacked vertically
- **ROOT**: Contains details on request, including date/time
- **DEFINE** or **STANDARD** or **TERMINOLOGY**: Contains properties of the object
- **DATASETS**: Contains the content from the Datasets tab
- **VARIABLES**: Contains the content from the Variables tab
- **VALUE_LEVEL**: Contains the content from the Value Level tab
- **METHODS** and **COMMENTS**: Contains the content on the Methods and Comments tabs
- **DICTIONARIES**: Contains the content from the Dictionaries table
- **DOCUMENTS**: Contains the content from the Documents table
- **CODELISTS** and **TERMS**: Contains the codelist and terms metadata
- **ANALYSIS_DISPLAYS** and **ANALYSIS_RESULTS**: Contain the content for ARM

In summary, every column visible to you in the P21E application is a part of the response (see Display 2 for an example).

	ordinal_root	ordinal_variables	order	dataset	variable	label	dataType	length	significantDigits	f
1	1	1	1	ADAE	STUDYID	Study Identifier	text	14		
2	1	2	2	ADAE	USUBJID	Unique Subject Identifier	text	22		
3	1	3	3	ADAE	SUBJID	Subject Identifier for the Study	text	7		
4	1	4	4	ADAE	SITEID	Study Site Identifier	text	3		
5	1	5	5	ADAE	AGE	Age	integer	2		
6	1	6	6	ADAE	AGEU	Age Units	text	5		
7	1	7	7	ADAE	AGEGR1	Pooled Age Group 1	text	13		
8	1	8	8	ADAE	AGEGR1N	Pooled Age Group 1 (N)	integer	1		
9	1	9	9	ADAE	SEX	Sex	text	1		
10	1	10	10	ADAE	RACE	Race	text	5		
11	1	11	11	ADAE	COUNTRY	Country	text	3		
12	1	12	12	ADAE	TRTP	Planned Treatment	text	33		
13	1	13	13	ADAE	TRTA	Actual Treatment	text	33		
14	1	14	14	ADAE	TRTSDT	Date of First Exposure to Treatment	integer	5		DATE
15	1	15	15	ADAE	TRTSDTM	Datetime of First Exposure to Treatment	integer	10		DATE
16	1	16	16	ADAE	TRTEDT	Date of Last Exposure to Treatment	integer	5		DATE
17	1	17	17	ADAE	TRTEDTM	Datetime of Last Exposure to Treatment	integer	10		DATE
18	1	18	18	ADAE	COMPLDT	Date of Study Completion	integer	5		DATE
19	1	19	19	ADAE	COMPLDT	Datetime of Study Completion	integer	10		DATE
20	1	20	20	ADAE	PAHDIAG	Current PAH Diagnosis	text	41		
21	1	21	21	ADAE	PAHGR1	Pooled Duration of PAH Group 1	text	11		
22	1	22	22	ADAE	PAHGR1N	Pooled Duration of PAH Group 1 (N)	integer	1		
23	1	23	23	ADAE	ENRLFL	Enrolled Population Flag	text	1		
24	1	24	24	ADAE	SAFFL	Safety Population Flag	text	1		
25	1	25	25	ADAE	COMPLFL	Completers Population Flag	text	1		

Display 2. Example of metadata from the WORK.VARIABLES table

EXAMPLES AND IDEAS FOR IMPLEMENTATION

This metadata can be used in all sorts of ways to aid in programming and code development. One example is to ensure the dataset metadata matches the spec and resulting define.xml to avoid validation issues downstream. Consider the scenario where we set the dataset label based on the value found in the spec:

```
* Assign dataset label to macro variable;
data _null_;
 set datasets;
  call symput(catx("_",dataset,"label"),label);
run;
data adsl(label="&adsl_label");
 set adsl;
run;
```

Consider tasks like setting up the dataset for finalization—defining things like KEEP variables or DROP variables. This can be data driven based on the content in the P21E spec. Alternatively, we can also set up the variable attributes based on the content defined in P21E so that the dataset matches the spec perfectly.

```
* Step through the variables to build an attrib statement;
data _null_;
 set variables;
  where dataset="ADSL";
   by dataset ordinal_variables order;
    length attrib $10000;
    retain attrib;

    if first.dataset then do;
     attrib="attrib ";
    end;

    attrib = strip(attrib) || " " || upcase(variable) ||
         " label='" || strip(label) || "' length=" ||
         ifc(strip(upcase(dataType)) in("INTEGER","FLOAT"),"8.","$"
         || strip(put(length,best.)));

    if last.dataset then do;
     attrib = strip(attrib) || ";";
     call symput('setAttribs',attrib);
    end;
run;

data adsl;
 &setAttribs;
 set adsl;
run;
```

Another idea is using the code and decoded term metadata to help define variable pairs using the content from the spec directly. This ensures that the coded variable (--TESTCD or --PARAMCD) and decoded variable (--TEST or --PARAM) matches the codelist exactly. To do this, you can merge the coded variable values against the WORK.TERMS datasets and set the value of the decoded variable easily. This eliminates the problem of having trivial minor differences in your data values vs. what is found in the define.xml document. You could even make a **%visitnum** macro that assigns VISIT based on the value of VISITNUM using the content found in P21E as the source.

If set up correctly, every time you run your program and the P21E REST API request is made, the content returned from P21E will be updated and reflected in your programming. Tweaks made to variable labels, types, lengths, and even codelists are reflected the next time the SAS program is executed.

CONCLUSION

The P21E REST API helps automate tasks that customers currently access through the P21E user interface. It can help to streamline processes and reduce the time it takes to perform these jobs. Spec-driven programming is not a new concept; it's been around for some time. However, our REST API helps make this a closer reality for our customers who still rely on Microsoft Excel and PROC IMPORT to accomplish this. Using our REST API for a spec-driven programming approach will also help to reduce the number of validation findings that are nothing more than a headache—issues like DD0067 (*Variable is not referenced*), SD1324 (*Define.xml/dataset variable label mismatch*), and CT2001/CT2002 (*Variable value not found in codelist*) while helping to ensure the spec remains the single source of truth.

I look forward to hearing and seeing all the unique implementation ideas that our customers create with P21E's REST API. Please share using my contact information below.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Trevor Mankus
trevor.mankus@certara.com