Developer Manual

DHIS core version 2.35

DHIS2 Documentation Team

dhis2

Last update: 2021-06-14

Table of contents

Web API

The Web API is a component which makes it possible for external systems to access and manipulate data stored in an instance of DHIS2. More precisely, it provides a programmatic interface to a wide range of exposed data and service methods for applications such as third-party software clients, web portals and internal DHIS2 modules.

Introduction

The Web API adheres to many of the principles behind the REST architectural style. To mention some few and important ones:

1. The fundamental building blocks are referred to as *resources*. A resource can be anything exposed to the Web, from a document to a business process - anything a client might want to interact with. The information aspects of a resource can be retrieved or exchanged through resource *representations*. A representation is a view of a resource's state at any given time. For instance, the *reportTable* resource in DHIS2 represents a tabular report of aggregated data for a certain set of parameters. This resource can be retrieved in a variety of representation formats including HTML, PDF, and MS Excel.

2. All resources can be uniquely identified by a *URI* (also referred to as *URL*). All resources have a default representation. You can indicate that you are interested in a specific representation by supplying an *Accept* HTTP header, a file extension or a *format* query parameter. So in order to retrieve the PDF representation of a report table you can supply an *Accept: application/pdf* header or append *.pdf* or *?format=pdf* to your request URL.

3. Interactions with the API requires the correct use of HTTP *methods* or *verbs*. This implies that for a resource you must issue a *GET* request when you want to retrieve it, *POST* request when you want to create one, *PUT* when you want to update it and *DELETE* when you want to remove it. So if you want to retrieve the default representation of a report table you can send a GET request to e.g. */reportTable/iu8j/hYgF6t*, where the last part is the report table identifier.

4. Resource representations are *linkable*, meaning that representations advertise other resources which are relevant to the current one by embedding links into itself (please be aware that you need to request *href* in your field filter to have this working. This feature greatly improves the usability and robustness of the API as we will see later. For instance, you can easily navigate to the indicators which are associated with a report table from the *reportTable* resource through the embedded links using your preferred representation format.

While all of this might sound complicated, the Web API is actually very simple to use. We will proceed with a few practical examples in a minute.

Authentication

The DHIS2 Web API supports two protocols for authentication, Basic Authentication and OAuth 2. You can verify and get information about the currently authenticated user by making a GET request to the following URL:

```
/api/33/me
```

And more information about authorities (and if a user has a certain authority) by using the endpoints:

```
/api/33/me/authorities
/api/33/me/authorities/ALL
```

Basic Authentication

The DHIS2 Web API supports *Basic authentication*. Basic authentication is a technique for clients to send login credentials over HTTP to a web server. Technically speaking, the username is appended with a colon and the password, Base64-encoded, prefixed Basic and supplied as the value of the *Authorization* HTTP header. More formally that is:

```
Authorization: Basic base64encode(username:password)
```

Most network-aware development environments provide support for Basic authentication, such as *Apache HttpClient* and *Spring RestTemplate*. An important note is that this authentication scheme provides no security since the username and password are sent in plain text and can be easily observed by an attacker. Using Basic is recommended only if the server is using SSL/TLS (HTTPS) to encrypt communication with clients. Consider this a hard requirement in order to provide secure interactions with the Web API.

Two-factor authentication

DHIS2 supports two-factor authentication. This can be enabled per user. When enabled, users will be asked to enter a 2FA code when logging in. You can read more about 2FA here.

OAuth2

DHIS2 supports the *OAuth2* authentication protocol. OAuth2 is an open standard for authorization which allows third-party clients to connect on behalf of a DHIS2 user and get a reusable *bearer token* for subsequent requests to the Web API. DHIS2 does not support fine-grained OAuth2 roles but rather provides applications access based on user roles of the DHIS2 user.

Each client for which you want to allow OAuth 2 authentication must be registered in DHIS2. To add a new OAuth2 client go to `Apps > Settings > OAuth2 Clients` in the user interface, click *Add new* and enter the desired client name and the grant types.

Adding a client using the Web API

An OAuth2 client can be added through the Web API. As an example, we can send a payload like this:

```
{
    "name": "OAuth2 Demo Client",
    "cid": "demo",
    "secret": "1e6db50c-0fee-11e5-98d0-3c15c2c6caf6",
    "grantTypes": ["password", "refresh_token", "authorization_code"],
    "redirectUris": ["http://www.example.org"]
}
```

The payload can be sent with the following command:

```
SERVER="https://play.dhis2.org/dev"
curl -X POST -H "Content-Type: application/json" -d @client.json
  -u admin:district "$SERVER/api/oAuth2Clients"
```

We will use this client as the basis for our next grant type examples.

Grant type password

The simplest of all grant types is the *password* grant type. This grant type is similar to basic authentication in the sense that it requires the client to collect the user's username and password. As an example we can use our demo server:

```
SERVER="https://play.dhis2.org/dev"
SECRET="1e6db50c-0fee-11e5-98d0-3c15c2c6caf6"

curl -X POST -H "Accept: application/json" -u demo:$SECRET "$SERVER/uaa/oauth/token"
  -d grant_type=password -d username=admin -d password=district
```

This will give you a response similar to this:

```
{
    "expires_in": 43175,
    "scope": "ALL",
    "access_token": "07fc551c-806c-41a4-9a8c-10658bd15435",
    "refresh_token": "a4e4de45-4743-481d-9345-2cfe34732fcc",
    "token_type": "bearer"
}
```

For now, we will concentrate on the `access_token`, which is what we will use as our authentication (bearer) token. As an example, we will get all data elements using our token:

```
SERVER="https://play.dhis2.org/dev"
curl -H "Authorization: Bearer 07fc551c-806c-41a4-9a8c-10658bd15435" "$SERVER/api/33/
dataElements.json"
```

Grant type refresh_token

In general the access tokens have limited validity. You can have a look at the `expires_in` property of the response in the previous example to understand when a token expires. To get a fresh `access_token` you can make another round trip to the server and use `refresh_token` which allows you to get an updated token without needing to ask for the user credentials one more time.

```
SERVER="https://play.dhis2.org/dev"
SECRET="1e6db50c-0fee-11e5-98d0-3c15c2c6caf6"
REFRESH_TOKEN="a4e4de45-4743-481d-9345-2cfe34732fcc"

curl -X POST -H "Accept: application/json" -u demo:$SECRET "$SERVER/uaa/oauth/token"
  -d "grant_type=refresh_token" -d "refresh_token=$REFRESH_TOKEN"
```

The response will be exactly the same as when you get a token to start with.

Grant type authorization_code

Authorized code grant type is the recommended approach if you don't want to store the user credentials externally. It allows DHIS2 to collect the username/password directly from the user instead of the client collecting them and then authenticating on behalf of the user. Please be aware that this approach uses the `redirectUris` part of the client payload.

Step 1: Visit the following URL using a web browser. If you have more than one redirect URIs, you might want to add `&redirect_uri=http://www.example.org` to the URL:

```
SERVER="https://play.dhis2.org/dev"
$SERVER/uaa/oauth/authorize?client_id=demo&response_type=code
```

Step 2: After the user has successfully logged in and accepted your client access, it will redirect back to your redirect uri like this:

```
http://www.example.org/?code=XYZ
```

Step 3: This step is similar to what we did in the password grant type, using the given code, we will now ask for an access token:

```
SERVER="https://play.dhis2.org/dev"
SECRET="1e6db50c-0fee-11e5-98d0-3c15c2c6caf6"

curl -X POST -u demo:$SECRET -H "Accept: application/json" $SERVER/uaa/oauth/token
-d "grant_type=authorization_code" -d "code=XYZ"
```

Error and info messages

The Web API uses a consistent format for all error/warning and informational messages:

```
{
    "httpStatus": "Forbidden",
    "message": "You don't have the proper permissions to read objects of this type.",
    "httpStatusCode": 403,
    "status": "ERROR"
}
```

Here we can see from the message that the user tried to access a resource I did not have access to. It uses the http status code 403, the http status message *forbidden* and a descriptive message.

WebMessage properties

Name	Description
httpStatus	HTTP Status message for this response, see RFC 2616 (Section 10) for more information.
httpStatus Code	HTTP Status code for this response, see RFC 2616 (Section 10) for more information.

Name	Description
status	DHIS2 status, possible values are *OK* | *WARNING* | *ERROR*, where `OK` means everything was successful, `ERROR` means that operation did not complete and `WARNING` means the operation was partially successful, if the message contains a `response` property, please look there for more information.
message	A user-friendly message telling whether the operation was a success or not.
devMessage	A more technical, developer-friendly message (not currently in use).
response	Extension point for future extension to the WebMessage format. This will be documented when it starts being used.

Date and period format

Throughout the Web API, we refer to dates and periods. The date format is:

```
yyyy-MM-dd
```

For instance, if you want to express March 20, 2014, you must use *2014-03-20*.

The period format is described in the following table (also available on the API endpoint `/api/periodTypes`)

Period format

Interval	Format	Example	Description
Day	*yyyyMMdd*	20040315	March 15, 2004
Week	*yyyy*W*n*	2004W10	Week 10 2004
Week Wednesday	*yyyy*WedW*n*	2015WedW5	Week 5 with start Wednesday
Week Thursday	*yyyy*ThuW*n*	2015ThuW6	Week 6 with start Thursday
Week Saturday	*yyyy*SatW*n*	2015SatW7	Week 7 with start Saturday
Week Sunday	*yyyy*SunW*n*	2015SunW8	Week 8 with start Sunday
Bi-week	*yyyy*BiW*n*	2015BiW1	Week 1-2 20015
Month	*yyyyMM*	200403	March 2004
Bi-month	*yyyyMM*B	200401B	January-February 2004
Quarter	*yyyy*Q*n*	2004Q1	January-March 2004
Six-month	*yyyy*S*n*	2004S1	January-June 2004
Six-month April	*yyyy*AprilSn	2004AprilS1	April-September 2004
Year	yyyy	2004	2004
Financial Year April	yyyyApril	2004April	Apr 2004-Mar 2005
Financial Year July	yyyyJuly	2004July	July 2004-June 2005
Financial Year Oct	yyyyOct	2004Oct	Oct 2004-Sep 2005

Relative Periods

In some parts of the API, like for the analytics resource, you can utilize relative periods in addition to fixed periods (defined above). The relative periods are relative to the current date and allow e.g. for creating dynamic reports. The available relative period values are:

```
THIS_WEEK, LAST_WEEK, LAST_4_WEEKS, LAST_12_WEEKS, LAST_52_WEEKS,
THIS_MONTH, LAST_MONTH, THIS_BIMONTH, LAST_BIMONTH, THIS_QUARTER, LAST_QUARTER,
THIS_SIX_MONTH, LAST_SIX_MONTH, MONTHS_THIS_YEAR, QUARTERS_THIS_YEAR,
THIS_YEAR, MONTHS_LAST_YEAR, QUARTERS_LAST_YEAR, LAST_YEAR, LAST_5_YEARS, LAST_12_MONTHS,
LAST_3_MONTHS, LAST_6_BIMONTHS, LAST_4_QUARTERS, LAST_2_SIXMONTHS, THIS_FINANCIAL_YEAR,
LAST_FINANCIAL_YEAR, LAST_5_FINANCIAL_YEARS
```

Identifier schemes

This section provides an explanation of the identifier scheme concept. Identifier schemes are used to map metadata objects to other metadata during import, and to render metadata as part of exports. Please note that not all schemes work for all API calls, and not all schemes can be used for both input and output. This is outlined in the sections explaining the various Web APIs.

The full set of identifier scheme object types available are listed below, using the name of the property to use in queries:

- idScheme
- dataElementIdScheme
- categoryOptionComboIdScheme
- orgUnitIdScheme
- programIdScheme
- programStageIdScheme
- trackedEntityIdScheme
- trackedEntityAttributeIdScheme

The general idScheme applies to all types of objects. It can be overridden by specific object types.

The default scheme for all parameters is UID (stable DHIS2 identifiers). The supported identifier schemes are described in the table below.

Scheme Values

Scheme	Description
ID, UID	Match on DHIS2 stable Identifier, this is the default id scheme.
CODE	Match on DHIS2 Code, mainly used to exchange data with an external system.
NAME	Match on DHIS2 Name, please note that this uses what is available as *object.name*, and not the translated name. Also note that names are not always unique, and in that case, they can not be used.
ATTRIBUTE:ID	Match on metadata attribute, this attribute needs to be assigned to the type you are matching on, and also that the unique property is set to *true*. The main usage of this is also to exchange data with external systems, it has some advantages over *CODE* since multiple attributes can be added, so it can be used to synchronize with more than one system.

Note that identifier schemes is not an independent feature but needs to be used in combination with resources such as data value import and metadata import.

As an example, to specify CODE as the general id scheme and override with UID for organisation unit id scheme you can use these query parameters:

```
?idScheme=CODE&orgUnitIdScheme=UID
```

As another example, to specify an attribute for the organisation unit id scheme, code for the data element id scheme and use the default UID id scheme for all other objects you can use these parameters:

```
?orgUnitIdScheme=ATTRIBUTE:j38fk2dKFsG&dataElementIdScheme=CODE
```

Browsing the Web API

The entry point for browsing the Web API is `/api`. This resource provides links to all available resources. Four resource representation formats are consistently available for all resources: HTML, XML, JSON, and JSONP. Some resources will have other formats available, like MS Excel, PDF, CSV, and PNG. To explore the API from a web browser, navigate to the `/api` entry point and follow the links to your desired resource, for instance `/api/dataElements`. For all resources which return a list of elements certain query parameters can be used to modify the response:

Query parameters

Param	Option values	Default option	Description
paging	true | false	true	Indicates whether to return lists of elements in pages.
page	number	1	Defines which page number to return.
pageSize	number	50	Defines the number of elements to return for each page.
order	property:asc/iasc/desc/idesc		Order the output using a specified order, only properties that are both persisted and simple (no collections, idObjects etc) are supported. iasc and idesc are case insensitive sorting.

An example of how these parameters can be used to get a full list of data element groups in XML response format is:

```
/api/dataElementGroups.xml?links=false&paging=false
```

You can query for elements on the name property instead of returning a full list of elements using the *query* query variable. In this example we query for all data elements with the word "anaemia" in the name:

```
/api/dataElements?query=anaemia
```

You can get specific pages and page sizes of objects like this:

```
/api/dataElements.json?page=2&pageSize=20
```

You can completely disable paging like this:

```
/api/indicatorGroups.json?paging=false
```

To order the result based on a specific property:

```
/api/indicators.json?order=shortName:desc
```

You can find an object based on its ID across all object types through the *identifiableObjects* resource:

```
/api/identifiableObjects/<id>
```

Translation

DHIS2 supports translations of database content, such as data elements, indicators, and programs. All metadata objects in the Web API have properties meant to be used for display / UI purposes, which include *displayName*, *displayShortName* and *displayDescription*.

Translate options

Parameter	Values	Description
translate	true | false	Translate display* properties in metadata output (displayName, displayShortName, displayDescription, and displayFormName for data elements). Default value is true.
locale	Locale to use	Translate metadata output using a specified locale (requires translate=true).

Translation API

The translations for an object is rendered as part of the object itself in the *translations* array. Note that the *translations* array in the JSON/XML payloads is normally pre-filtered for you, which means they can not directly be used to import/export translations (as that would normally overwrite locales other than current users).

Example of data element with translation array filtered on user locale:

```
{
    "id": "FTRrcoaog83",
    "displayName": "Accute French",
    "translations": [
        {
            "property": "SHORT_NAME",
            "locale": "fr",
            "value": "Accute French"
        },
        {
            "property": "NAME",
            "locale": "fr",
            "value": "Accute French"
        }
    ]
}
```

Example of data element with translations turned off:

```
{
    "id": "FTRrcoaog83",
    "displayName": "Accute Flaccid Paralysis (Deaths < 5 yrs)",
    "translations": [
        {
            "property": "FORM_NAME",
            "locale": "en_FK",
            "value": "aa"
        },
        {
            "property": "SHORT_NAME",
            "locale": "en_GB",
            "value": "Accute Flaccid Paral"
        },
        {
            "property": "SHORT_NAME",
            "locale": "fr",
            "value": "Accute French"
        },
        {
            "property": "NAME",
            "locale": "fr",
            "value": "Accute French"
        },
        {
            "property": "NAME",
            "locale": "en_FK",
            "value": "aa"
        },
        {
            "property": "DESCRIPTION",
            "locale": "en_FK",
            "value": "aa"
        }
    ]
}
```

Note that even if you get the unfiltered result, and are using the appropriate type endpoint i..e `/api/dataElements` we do not allow updates, as it would be too easy to make mistakes and overwrite the other available locales.

To read and update translations you can use the special translations endpoint for each object resource. These can be accessed by *GET* or *PUT* on the appropriate `/api/<object-type>/<object-id>/translations` endpoint.

As an example, for a data element with identifier FTRrcoaog83, you could use `/api/dataElements/FTRrcoaog83/translations` to get and update translations. The fields available are `property` with options *NAME*, *SHORT_NAME*, *DESCRIPTION*, `locale` which supports any valid locale ID and the translated property `value`.

Example of NAME property for French locale:

```
{
    "property": "NAME",
    "locale": "fr",
    "value": "Paralysie Flasque Aiguë (Décès <5 ans)"
}
```

This payload would then be added to a translation array, and sent back to the appropriate endpoint:

```
{
    "translations": [
        {
            "property": "NAME",
            "locale": "fr",
            "value": "Paralysie Flasque Aiguë (Décès <5 ans)"
        }
    ]
}
```

For a data element with ID *FTRrcoaog83* you can *PUT* this to `/api/dataElements/FTRrcoaog83/translations`. Make sure to send all translations for the specific object and not just for a single locale (if not you will potentially overwrite existing locales for other locales).

Web API versions

The Web API is versioned starting from DHIS 2.25. The API versioning follows the DHIS2 major version numbering. As an example, the API version for DHIS 2.33 is 33.

You can access a specific API version by including the version number after the `/api` component, as an example like this:

```
/api/33/dataElements
```

If you omit the version part of the URL, the system will use the current API version. As an example, for DHIS 2.25, when omitting the API part, the system will use API version 25. When developing API clients it is recommended to use explicit API versions (rather than omitting the API version), as this will protect the client from unforeseen API changes.

The last three API versions will be supported. As an example, DHIS version 2.27 will support API version 27, 26 and 25.

Note that the metadata model is not versioned and that you might experience changes e.g. in associations between objects. These changes will be documented in the DHIS2 major version release notes.

Metadata object filter

To filter the metadata there are several filter operations that can be applied to the returned list of metadata. The format of the filter itself is straight-forward and follows the pattern *property:operator:value*, where *property* is the property on the metadata you want to filter on, *operator* is the comparison operator you want to perform and *value* is the value to check against (not all operators require value). Please see the *schema* section to discover which properties are available. Recursive filtering, ie. filtering on associated objects or collection of objects, is supported as well.

Available Operators

Operator	Types	Value required	Description
eq	string | boolean | integer | float | enum | collection (checks for size) | date	true	Equality
!eq	string | boolean | integer | float | enum | collection (checks for size) | date	true	Inequality
ne	string | boolean | integer | float | enum | collection (checks for size) | date	true	Inequality
like	string	true	Case sensitive string, match anywhere
!like	string	true	Case sensitive string, not match anywhere
\$like	string	true	Case sensitive string, match start
!\$like	string	true	Case sensitive string, not match start
like\$	string	true	Case sensitive string, match end
!like\$	string	true	Case sensitive string, not match end
ilike	string	true	Case insensitive string, match anywhere
!ilike	string	true	Case insensitive string, not match anywhere
\$ilike	string	true	Case insensitive string, match start
!\$ilike	string	true	Case insensitive string, not match start
ilike\$	string	true	Case insensitive string, match end
!ilike\$	string	true	Case insensitive string, not match end

Operator	Types	Value required	Description
gt	string | boolean | integer | float | collection (checks for size) | date	true	Greater than
ge	string | boolean | integer | float | collection (checks for size) | date	true	Greater than or equal
lt	string | boolean | integer | float | collection (checks for size) | date	true	Less than
le	string | boolean | integer | float | collection (checks for size) | date	true	Less than or equal
null	all	false	Property is null
!null	all	false	Property is not null
empty	collection	false	Collection is empty
token	string	true	Match on multiple tokens in search property
!token	string	true	Not match on multiple tokens in search property
in	string | boolean | integer | float | date	true	Find objects matching 1 or more values
!in	string | boolean | integer | float | date	true	Find objects not matching 1 or more values

Operators will be applied as logical *and* query, if you need a *or* query, you can have a look at our *in* filter (also have a look at the section below). The filtering mechanism allows for recursion. See below for some examples.

Get data elements with id property ID1 or ID2:

```
/api/dataElements?filter=id:eq:ID1&filter=id:eq:ID2
```

Get all data elements which have the dataSet with id ID1:

```
/api/dataElements?filter=dataSetElements.dataSet.id:eq:ID1
```

Get all data elements with aggregation operator "sum" and value type "int":

```
/api/dataElements.json?filter=aggregationOperator:eq:sum&filter=type:eq:int
```

You can do filtering within collections, e.g. to get data elements which are members of the "ANC" data element group you can use the following query using the id property of the associated data element groups:

```
/api/dataElements.json?filter=dataElementGroups.id:eq:qfxEYY9xAl6
```

Since all operators are *and* by default, you can't find a data element matching more than one id, for that purpose you can use the *in* operator.

```
/api/dataElements.json?filter=id:in:[fbfJHSPpUQD,cYeuwXTCPkU]
```

Logical operators

As mentioned in the section before, the default logical operator applied to the filters is *AND* which means that all object filters must be matched. There are however cases where you want to match on one of several filters (maybe id and code field) and in those cases, it is possible to switch the root logical operator from *AND* to *OR* using the *rootJunction* parameter.

Example: Normal filtering where both id and code must match to have a result returned

```
/api/dataElements.json?filter=id:in:[id1,id2]&filter=code:eq:code1
```

Example: Filtering where the logical operator has been switched to OR and now only one of the filters must match to have a result returned

```
/api/dataElements.json?filter=id:in:[id1,id2]&filter=code:eq:code1&rootJunction=OR
```

Identifiable token filter

In addition to the specific property based filtering mentioned above, we also have *token* based *AND* filtering across a set of properties: id, code, and name (also shortName if available). These properties are commonly referred to as *identifiable*. The idea is to filter metadata whose id, name, code or short name containing something.

Example: Filter all data elements containing 2^{nd} in any of the following: id,name,code, shortName

```
/api/dataElements.json?filter=identifiable:token:2nd
```

It is also possible to specify multiple filtering values.

Example: Get all data elements where *ANC visit* is found in any of the *identifiable* properties. The system returns all data elements where both tokens (ANC and visit) are found anywhere in identifiable properties.

```
/api/dataElements.json?filter=identifiable:token:ANC visit
```

It is also possible to combine the identifiable filter with property-based filter and expect the *rootJunction* to be applied.

```
/api/dataElements.json?filter=identifiable:token:ANC visit&filter=displayName:ilike:tt1

/api/dataElements.json?filter=identifiable:token:ANC visit
  &filter=displayName:ilike:tt1&rootJunction=OR
```

Capture Scope filter

In addition to the filtering mentioned above, we have a special filtering query parameter named *restrictToCaptureScope*. If *restrictToCaptureScope* is set to true, only those metadata objects that are either unassigned to any organisation units or those that are assigned explicitly to the logged in users capture scope org units will be returned in the response. In addition to filtering the metadata object lists, an additional filtering of the associated organisation units to only include the capture scoped organisation units will be done. This filtering parameter can be used for Program and CategoryOption listing APIs.

This feature is generally beneficial to reduce the payload size if there are large number of organisation units associated to various metadata objects.

Some examples

```
/api/categoryOptions.json?restrictToCaptureScope=true&fields=*

/api/programs.json?restrictToCaptureScope=true&fields=*
```

All existing filters will work in addition to the capture scope filter.

```
/api/categoryOptions.json?restrictToCaptureScope=true&fields=*&filter=displayName:ilike:11
```

Metadata field filter

In many situations, the default views of the metadata can be too verbose. A client might only need a few fields from each object and want to remove unnecessary fields from the response. To discover which fields are available for each object please see the *schema* section.

The format for include/exclude allows for infinite recursion. To filter at the "root" level you can just use the name of the field, i.e. `?fields=id,name` which would only display the `id` and `name` fields for every object. For objects that are either collections or complex objects with properties on their own, you can use the format `?fields=id,name,dataSets[id,name]` which would return `id`, `name` of the root, and the `id` and `name` of every data set on that object. Negation can be done with the exclamation operator, and we have a set of presets of field select. Both XML and JSON are supported.

Example: Get `id` and `name` on the indicators resource:

```
/api/indicators?fields=id,name
```

Example: Get `id` and `name` from dataElements, and `id` and `name` from the dataSets on dataElements:

```
/api/dataElements?fields=id,name,dataSets[id,name]
```

To exclude a field from the output you can use the exclamation ! operator. This is allowed anywhere in the query and will simply not include that property as it might have been inserted in some of the presets.

A few presets (selected fields groups) are available and can be applied using the : operator.

Property operators

Operator	Description
<field-name>	Include property with name, if it exists.
<object>[<field-name>, ...]	Includes a field within either a collection (will be applied to every object in that collection), or just on a single object.
!<field-name>, <object>[!<field-name>	Do not include this field name, it also works inside objects/collections. Useful when you use a preset to include fields.
*, <object>[*]	Include all fields on a certain object, if applied to a collection, it will include all fields on all objects on that collection.
:<preset>	Alias to select multiple fields. Three presets are currently available, see the table below for descriptions.

Field presets

Preset	Description
all	All fields of the object
*	Alias for all
identifiable	Includes id, name, code, created and lastUpdated fields
nameable	Includes id, name, shortName, code, description, created and lastUpdated fields
persisted	Returns all persisted property on an object, does not take into consideration if the object is the owner of the relation.
owner	Returns all persisted property on an object where the object is the owner of all properties, this payload can be used to update through the API.

Example: Include all fields from dataSets except organisationUnits:

```
/api/dataSets?fields=:all,!organisationUnits
```

Example: Include only id, name and the collection of organisation units from a data set, but exclude the id from organisation units:

```
/api/dataSets/BfMAe6Itzgt?fields=id,name,organisationUnits[:all,!id]
```

Example: Include nameable properties from all indicators:

```
/api/indicators.json?fields=:nameable
```

Field transformers

In DHIS2.17 we introduced field transformers, the idea is to allow further customization of the properties on the server-side.

```
/api/dataElements/ID?fields=id~rename(i),name~rename(n)
```

This will rename the *id* property to *i* and *name* property to *n*.

Multiple transformers can be used by repeating the transformer syntax:

```
/api/dataElementGroups.json?fields=id,displayName,dataElements~isNotEmpty~rename(haveDataElements)
```

Available Transformers

Name	Arguments	Description
size		Gives sizes of strings (length) and collections
isEmpty		Is string or collection empty
isNotEmpty		Is string or collection not empty
rename	Arg1: name	Renames the property name
paging	Arg1: page,Arg2: pageSize	Pages a collection, default pageSize is 50.
pluck	Optional Arg1: fieldName	Converts an array of objects to an array of a selected field of that object. By default, the first field that is returned by the collection is used (normally the ID).

Examples

Examples of transformer usage.

```
/api/dataElements?fields=dataSets~size

/api/dataElements?fields=dataSets~isEmpty

/api/dataElements?fields=dataSets~isNotEmpty

/api/dataElements/ID?fields=id~rename(i),name~rename(n)

/api/dataElementGroups?fields=id,displayName,dataElements~paging(1;20)

# Include array with IDs of organisation units:
/api/categoryOptions.json?fields=id,organisationUnits~pluck

# Include array with names of organisation units (collection only returns field name):
/api/categoryOptions.json?fields=id,organisationUnits~pluck[name]
```

Metadata create, read, update, delete, validate

All metadata entities in DHIS2 have their own API endpoint which supports *CRUD* operations (create, read, update and delete). The endpoint URLs follows this format:

```
/api/<entityName>
```

The *entityName* uses the camel-case notation. As an example, the endpoint for *data elements* is:

```
/api/dataElements
```

Create / update parameters

The following request query parameters are available across all metadata endpoints.

Available Query Filters

Param	Type	Required	Options (default first)	Description
preheatCache	boolean	false	true | false	Turn cache-map preheating on/off. This is on by default, turning this off will make initial load time for importer much shorter (but will make the import itself slower). This is mostly used for cases where you have a small XML/JSON file you want to import, and don't want to wait for cache-map preheating.
strategy	enum	false	CREATE_AND_UPDATE | CREATE | UPDATE | DELETE	Import strategy to use, see below for more information.
mergeMode	enum	false	REPLACE, MERGE	Strategy for merging of objects when doing updates. REPLACE will just overwrite the property with the new value provided, MERGE will only set the property if it is not null (only if the property was provided).

Creating and updating objects

For creating new objects you will need to know the endpoint, the type format, and make sure that you have the required authorities. As an example, we will create and update a *constant*. To figure

out the format, we can use the new *schema* endpoint for getting format description. So we will start with getting that info:

```
http://<server>/api/schemas/constant.json
```

From the output, you can see that the required authorities for create are `F_CONSTANT_ADD`, and the important properties are: *name* and *value*. From this, we can create a JSON payload and save it as a file called constant.json:

```
{
    "name": "PI",
    "value": "3.14159265359"
}
```

The same content as an XML payload:

```
<constant name="PI" xmlns="http://dhis2.org/schema/dxf/2.0">
  <value>3.14159265359</value>
</constant>
```

We are now ready to create the new *constant* by sending a POST request to the *constants*endpoint with the JSON payload using curl:

```
curl -d @constant.json "http://server/api/constants" -X POST
  -H "Content-Type: application/json" -u user:password
```

A specific example of posting the constant to the demo server:

```
curl -d @constant.json "https://play.dhis2.org/api/constants" -X POST
  -H "Content-Type: application/json" -u admin:district
```

If everything went well, you should see an output similar to:

```
{
    "status": "SUCCESS",
    "importCount": {
        "imported": 1,
        "updated": 0,
        "ignored": 0,
        "deleted": 0
    },
    "type": "Constant"
}
```

The process will be exactly the same for updating, you make your changes to the JSON/XML payload, find out the *ID* of the constant, and then send a PUT request to the endpoint including ID:

```
curl -X PUT -d @pi.json -H "Content-Type: application/json"
  -u user:password "http://server/api/constants/ID"
```

Deleting objects

Deleting objects is very straight forward, you will need to know the *ID* and the endpoint of the type you want to delete, let's continue our example from the last section and use a *constant*. Let's assume that the id is *abc123*, then all you need to do is the send the DELETE request to the endpoint + id:

```
curl -X DELETE -u user:password "http://server/api/constants/ID"
```

A successful delete should return HTTP status 204 (no content).

Adding and removing objects in collections

The collections resource lets you modify collections of objects.

Adding or removing single objects

In order to add or remove objects to or from a collection of objects you can use the following pattern:

```
/api/{collection-object}/{collection-object-id}/{collection-name}/{object-id}
```

You should use the POST method to add, and the DELETE method to remove an object. When there is a many-to-many relationship between objects, you must first determine which object owns the relationship. If it isn't clear which object this is, try the call both ways to see which works.

The components of the pattern are:

- collection object: The type of objects that owns the collection you want to modify.
- collection object id: The identifier of the object that owns the collection you want to modify.
- collection name: The name of the collection you want to modify.
- object id: The identifier of the object you want to add or remove from the collection.

As an example, in order to remove a data element with identifier IDB from a data element group with identifier IDA you can do a DELETE request:

```
DELETE /api/dataElementGroups/IDA/dataElements/IDB
```

To add a category option with identifier IDB to a category with identifier IDA you can do a POST request:

```
POST /api/categories/IDA/categoryOptions/IDB
```

Adding or removing multiple objects

You can add or remove multiple objects from a collection in one request with a payload like this:

```
{
    "identifiableObjects": [
        {
            "id": "IDA"
        },
        {
            "id": "IDB"
        },
        {
            "id": "IDC"
        }
    ]
}
```

Using this payload you can add, replace or delete items:

Adding Items:

```
POST /api/categories/IDA/categoryOptions
```

Replacing Items:

```
PUT /api/categories/IDA/categoryOptions
```

Delete Items:

```
DELETE /api/categories/IDA/categoryOptions
```

Adding and removing objects in a single request

You can both add and remove objects from a collection in a single POST request to the following URL:

```
POST /api/categories/IDA/categoryOptions
```

The payload format is:

```
{
    "additions": [
        {
            "id": "IDA"
        },
        {
            "id": "IDB"
        },
        {
            "id": "IDC"
        }
    ],
    "deletions": [
        {
            "id": "IDD"
        },
```

```
        {
            "id": "IDE"
        },
        {
            "id": "IDF"
        }
    ]
}
```

Validating payloads

DHIS 2 supports system wide validation of metadata payloads, which means that create and update operations on the API endpoints will be checked for valid payload before allowing changes to be made. To find out what validations are in place for a specific endpoint, have a look at the `/api/schemas` endpoint, i.e. to figure out which constraints a data element have, you would go to `/api/schemas/dataElement`.

You can also validate your payload manually by sending it to the proper schema endpoint. If you wanted to validate the constant from the create section before, you would send it like this:

```
POST /api/schemas/constant
```

A simple (non-validating) example would be:

```
curl -X POST -d "{\"name\": \"some name\"}" -H "Content-Type: application/json"
  -u admin:district "https://play.dhis2.org/dev/api/schemas/dataElement"
```

Which would yield the result:

```
[
    {
        "message": "Required property missing.",
        "property": "type"
    },
    {
        "property": "aggregationOperator",
        "message": "Required property missing."
    },
    {
        "property": "domainType",
        "message": "Required property missing."
    },
    {
        "property": "shortName",
        "message": "Required property missing."
    }
]
```

Partial updates

For cases where you don't want or need to update all properties on a object (which means downloading a potentially huge payload, change one property, then upload again) we now support partial update, for one or more properties.

The payload for doing partial updates are the same as when you are doing a full update, the only difference is that you only include the properties you want to update, i.e.:

```
{
    "name": "Updated Name",
    "zeroIsSignificant": true
}
```

An example curl command looks like this:

```
curl -X PATCH -d @file.json -H "Content-Type: application/json"
  -u admin:district "https://play.dhis2.org/dev/api/dataElements/fbfJHSPpUQD"
```

Metadata export

This section explains the metatada API which is available at `/api/metadata`. XML and JSON resource representations are supported.

```
/api/metadata
```

The most common parameters are described below in the "Export Parameter" table. You can also apply this to all available types by using `type:fields=<filter>` and `type:filter=<filter>`. You can also enable/disable the export of certain types by setting `type=true|false`.

Export Parameter

Name	Options	Description
fields	Same as metadata field filter	Default field filter to apply for all types, default is `:owner`.
filter	Same as metadata object filter	Default object filter to apply for all types, default is `none`.
order	Same as metadata order	Default order to apply to all types, default is `name` if available, or `created` if not.
translate	false/true	Enable translations. Be aware that this is turned off by default (in other endpoints this is on by default).
locale	<locale>	Change from user locale, to your own custom locale.
defaults	INCLUDE/ EXCLUDE	Should auto-generated category object be included or not in the payload. If you are moving metadata between 2 non-synced instances, it might make sense to set this to EXCLUDE to ease the handling of these generated objects.
skipSharing	false/true	Enabling this will strip the sharing properties from the exported objects. This includes *user*, *publicAccess*, *userGroupAccesses*, *userAccesses*, and *externalAccess*.
download	false/true	Enabling this will add HTTP header Content-Disposition that specifies that the data should be handled as an attachment and will be offered by web browsers as a download.

Metadata export examples

Export all metadata. Be careful as the response might be very large depending on your metadata configuration:

```
/api/metadata
```

Export all metadata ordered by lastUpdated descending:

```
/api/metadata?defaultOrder=lastUpdated:desc
```

Export metadata only including indicators and indicator groups:

```
/api/metadata?indicators=true&indicatorGroups=true
```

Export id and displayName for all data elements, ordered by displayName:

```
/api/metadata?dataElements:fields=id,name&dataElements:order=displayName:desc
```

Export data elements and indicators where name starts with "ANC":

```
/api/metadata?filter=name:^like:ANC&dataElements=true&indicators=true
```

Metadata export with dependencies

When you want to exchange metadata for a data set, program or category combo from one DHIS2 instance to another instance there are three dedicated endpoints available:

```
/api/dataSets/{id}/metadata.json

/api/programs/{id}/metadata.json

/api/categoryCombos/{id}/metadata.json

/api/dashboards/{id}/metadata.json
```

These exports can then be imported using `/api/metadata`.

These endpoints also support the following parameters:

Export Parameter

Name	Options	Description
skipSharing	false/true	Enabling this will strip the sharing properties from the exported objects. This includes *user*, *publicAccess*, *userGroupAccesses*, *userAccesses*, and *externalAccess*.
download	false/true	Enabling this will add HTTP header Content-Disposition that specifies that the data should be handled as an attachment and will be offered by web browsers as a download.

Metadata import

This section explains the metadata import API. XML and JSON resource representations are supported. Metadata can be imported using a *POST* request.

```
/api/metadata
```

The importer allows you to import metadata payloads which may include many different entities and any number of objects per entity. The metadata export generated by the metadata export API can be imported directly.

The metadata import endpoint support a variety of parameters, which are listed below.

Import Parameter

Name	Options (first is default)	Description
importMode	COMMIT, VALIDATE	Sets overall import mode, decides whether or not to only `VALIDATE` or also `COMMIT` the metadata, this has similar functionality as our old dryRun flag.
identifier	UID, CODE, AUTO	Sets the identifier scheme to use for reference matching. `AUTO` means try `UID` first, then `CODE`.
importReportMode	ERRORS, FULL, DEBUG	Sets the `ImportReport` mode, controls how much is reported back after the import is done. `ERRORS` only includes *ObjectReports* for object which has errors. `FULL` returns an *ObjectReport* for all objects imported, and `DEBUG` returns the same plus a name for the object (if available).
preheatMode	REFERENCE, ALL, NONE	Sets the preheater mode, used to signal if preheating should be done for `ALL` (as it was before with *preheatCache=true*) or do a more intelligent scan of the objects to see what to preheat (now the default), setting this to `NONE` is not recommended.
importStrategy	CREATE_AND_UPDATE, CREATE, UPDATE, DELETE	Sets import strategy, `CREATE_AND_UPDATE` will try and match on identifier, if it doesn't exist, it will create the object.
atomicMode	ALL, NONE	Sets atomic mode, in the old importer we always did a *best effort* import, which means that even if some references did not exist, we would still import (i.e. missing data elements on a data element group import). Default for new importer is to not allow this, and similar reject any validation errors. Setting the `NONE` mode emulated the old behavior.
mergeMode	REPLACE, MERGE	Sets the merge mode, when doing updates we have two ways of merging the old object with the new one, `MERGE` mode will only overwrite the old property if the new one is not-null, for `REPLACE` mode all properties are overwritten regardless of null or not.
flushMode	AUTO, OBJECT	Sets the flush mode, which controls when to flush the internal cache. It is *strongly* recommended to keep this to `AUTO` (which is the default). Only use `OBJECT` for debugging purposes, where you are seeing hibernate exceptions and want to pinpoint the exact place where the stack happens (hibernate will only throw when flushing, so it can be hard to know which object had issues).

Name	Options (first is default)	Description
skipSharing	false, true	Skip sharing properties, does not merge sharing when doing updates, and does not add user group access when creating new objects.
skipValidation	false, true	Skip validation for import. `NOT RECOMMENDED`.
async	false, true	Asynchronous import, returns immediately with a *Location* header pointing to the location of the *importReport*. The payload also contains a json object of the job created.
inclusionStrategy	NON_NULL, ALWAYS, NON_EMPTY	*NON_NULL* includes properties which are not null, *ALWAYS* include all properties, *NON_EMPTY* includes non empty properties (will not include strings of 0 length, collections of size 0, etc.)
userOverrideMode	NONE, CURRENT, SELECTED	Allows you to override the user property of every object you are importing, the options are NONE (do nothing), CURRENT (use import user), SELECTED (select a specific user using overrideUser=X)
overrideUser	User ID	If userOverrideMode is SELECTED, use this parameter to select the user you want override with.

An example of a metadata payload to be imported looks like this. Note how each entity type have their own property with an array of objects:

```
{
    "dataElements": [
        {
            "name": "EPI - IPV 3 doses given",
            "shortName": "EPI - IPV 3 doses given",
            "aggregationType": "SUM",
            "domainType": "AGGREGATE",
            "valueType": "INTEGER_ZERO_OR_POSITIVE"
        },
        {
            "name": "EPI - IPV 4 doses given",
            "shortName": "EPI - IPV 4 doses given",
            "aggregationType": "SUM",
            "domainType": "AGGREGATE",
            "valueType": "INTEGER_ZERO_OR_POSITIVE"
        }
    ],
    "indicators": [
        {
            "name": "EPI - ADS stock used",
            "shortName": "ADS stock used",
            "numerator": "#{LTb8XeeqeqI}+#{Fs28ZQJET6V}-#{A3mHIZd2tPg}",
            "numeratorDescription": "ADS 0.05 ml used",
            "denominator": "1",
            "denominatorDescription": "1",
            "annualized": false,
            "indicatorType": {
                "id": "kHy61PbChXr"
            }
        }
    ]
}
```

When posting this payload to the metadata endpoint, the response will contain information about the parameters used during the import and a summary per entity type including how many objects were created, updated, deleted and ignored:


```
{
    "importParams": {
        "userOverrideMode": "NONE",
        "importMode": "COMMIT",
        "identifier": "UID",
        "preheatMode": "REFERENCE",
        "importStrategy": "CREATE_AND_UPDATE",
        "atomicMode": "ALL",
        "mergeMode": "REPLACE",
        "flushMode": "AUTO",
        "skipSharing": false,
        "skipTranslation": false,
        "skipValidation": false,
        "metadataSyncImport": false,
        "firstRowIsHeader": true,
        "username": "UNICEF_admin"
    },
    "status": "OK",
    "typeReports": [
        {
            "klass": "org.hisp.dhis.dataelement.DataElement",
            "stats": {
                "created": 2,
                "updated": 0,
                "deleted": 0,
                "ignored": 0,
                "total": 2
            }
        },
        {
            "klass": "org.hisp.dhis.indicator.Indicator",
            "stats": {
                "created": 1,
                "updated": 0,
                "deleted": 0,
                "ignored": 0,
                "total": 1
            }
        }
    ],
    "stats": {
        "created": 3,
        "updated": 0,
        "deleted": 0,
        "ignored": 0,
        "total": 3
    }
}
```


Render type (Experimental)

Some metadata types have a property named *renderType*. The render type property is a map between a *device* and a *renderingType*. Applications can use this information as a hint on how the object should be rendered on a specific device. For example, a mobile device might want to render a data element differently than a desktop computer.

There is currently two different kinds of renderingTypes available:

1. Value type rendering
2. Program stage section rendering

There is also 2 device types available:

1. MOBILE
2. DESKTOP

The following table lists the metadata and rendering types available. The value type rendering has addition constraints based on the metadata configuration, which will be shown in a second table.

Metadata and RenderingType overview

Metadata type	Available RenderingTypes
Program Stage Section	• LISTING (default) • SEQUENTIAL • MATRIX
Data element	• DEFAULT • DROPDOWN • VERTICAL_RADIOBUTTONS • HORIZONTAL_RADIOBUTTONS • VERTICAL_CHECKBOXES • HORIZONTAL_CHECKBOXES • SHARED_HEADER_RADIOBUTTONS • ICONS_AS_BUTTONS • SPINNER • ICON • TOGGLE • VALUE • SLIDER • LINEAR_SCALE

Since handling the default rendering of data elements and tracked entity attributes are depending on the value type of the object, there is also a DEFAULT type to tell the client it should be handled as normal. Program Stage Section is LISTING as default.

RenderingTypes allowed based on value types

Value type	Is object an optionset?	RenderingTypes allowed
TRUE_ONLY	No	DEFAULT, VERTICAL_RADIOBUTTONS, HORIZONTAL_RADIOBUTTONS, VERTICAL_CHECKBOXES, HORIZONTAL_CHECKBOXES, TOGGLE
BOOLEAN	No	
-	Yes	DEFAULT, DROPDOWN, VERTICAL_RADIOBUTTONS, HORIZONTAL_RADIOBUTTONS, VERTICAL_CHECKBOXES, HORIZONTAL_CHECKBOXES, SHARED_HEADER_RADIOBUTTONS, ICONS_AS_BUTTONS, SPINNER, ICON
INTEGER	No	DEFAULT, VALUE, SLIDER, LINEAR_SCALE, SPINNER
INTEGER_POSITIVE	No	
INTEGER_NEGATIVE	No	
INTEGER_ZERO_OR_POSITIVE	No	
NUMBER	No	
UNIT_INTERVAL	No	
PERCENTAGE	No	

A complete reference of the previous table can also be retrieved using the following endpoint:

```
GET /api/staticConfiguration/renderingOptions
```

Value type rendering also has some additional properties that can be set, which is usually needed when rendering some of the specific types:

renderType object properties

Property	Description	Type
type	The RenderingType of the object, as seen in the first table. This property is the same for both value type and program stage section, but is the only property available for program stage section.	Enum (See list in the Metadata and Rendering Type table)
min	Only for value type rendering. Represents the minimum value this field can have.	Integer
max	Only for value type rendering. Represents the maximum value this field can have.	Integer

Property	Description	Type
step	Only for value type rendering. Represents the size of the steps the value should increase, for example for SLIDER og LINEAR_SCALE	Integer
decimalPoints	Only for value type rendering. Represents the number of decimal points the value should use.	Integer

The *renderingType* can be set when creating or updating the metadata listed in the first table. An example payload for the rendering type for program stage section looks like this:

```
{
    "renderingType": {
        "type": "MATRIX"
    }
}
```

For data element and tracked entity attribute:

```
{
    "renderingType": {
        "type": "SLIDER",
        "min": 0,
        "max": 1000,
        "step": 50,
        "decimalPoints": 0
    }
}
```

Object Style

Most metadata have a property names "style". This property can be used by clients to represent the object in a certain way. The properties currently supported by style is as follows:

Style properties

Property	Description	Type
color	A color, represented by a hexadecimal.	String (#000000)
icon	An icon, represented by a icon-name.	String

Currently, there is no official list or support for icon-libraries, so this is currently up to the client to provide. The following list shows all objects that support style:

- Data element
- Data element category option
- Data set

Indicator
- Option
- Program
- Program Indicator
- Program Section
- Program Stage
- Program Stage Section
- Relationship (Tracker)
- Tracked Entity Attribute
- Tracked Entity Type

When creating or updating any of these objects, you can include the following payload to change the style:

```
{
    "style": {
        "color": "#ffffff",
        "icon": "my-beautiful-icon"
    }
}
```

ActiveMQ Artemis / AMQP 1.0 integration

By default DHIS2 will start up an embedded instance of ActiveMQ Artemis when the instance is booting up. For most use-cases, you do not need to configure anything to make use of this, but if your infrastructure have an existing AMQP 1.0 compliant service you want to use, you can change the defaults in your *dhis.conf* file using the keys in the table down below.

AMQP Configuration Keys

Key	Value (default first)	Description
amqp.mode	`EMBEDDED` | `NATIVE`	The default `EMBEDDED` starts up an internal AMQP service when the DHIS2 instance is starting up. If you want to connect to an external AMQP service you need to set the mode to `NATIVE`.
amqp.host	`127.0.0.1`	Host to bind to.
amqp.port	`15672`	If mode is `EMBEDDED` then start the embedded server on this port, if `NATIVE` then the client will use this port to connect to.
amqp.username	`guest`	Username to connect to if using `NATIVE` mode.
amqp.password	`guest`	Password to connect to if using `NATIVE` mode.
amqp.embedded.persistence	`false` | `true`	If mode is `EMBEDDED`, this property controls persistence of the internal queue.

CSV metadata import

DHIS2 supports import of metadata in the CSV format, such as data elements, organisation units and validation rules. Properties for the various metadata objects are identified based on the column order/column index (see below for details). You can omit non-required object properties/columns, but since the column order is significant, an empty column must be included. In other words, if you would like to specify properties/columns which appear late in the column order but not specify certain columns which appear early in the order you can include empty/blank columns for them.

The first row of the CSV file is considered to be a header and is ignored during import. The *comma* character should be used as a text delimiter. Text which contains commas must be enclosed in *double quotes*.

To upload metadata in CSV format you can make a POST request to the metadata endpoint:

```
POST /api/metadata?classKey=CLASS-KEY
```

The following object types are supported. The `classKey` query parameter is mandatory and can be found next to each object type in the table below.

Object types and keys

Object type	Class key
Data elements	DATA_ELEMENT
Data element groups	DATA_ELEMENT_GROUP
Category options	CATEGORY_OPTION
Category option groups	CATEGORY_OPTION_GROUP
Organisation units	ORGANISATION_UNIT
Organisation unit groups	ORGANISATION_UNIT_GROUP
Validation rules	VALIDATION_RULE
Option sets	OPTION_SET
Translations	TRANSLATION

> **Tip**
>
> If using *curl*, the `--data-binary` option should be used as it preserves line breaks and newlines, which is essential for CSV data.

As an example, to upload a file of data elements in CSV format with `curl` you can use the following command:

```
curl --data-binary @data_elements.csv "http://localhost/api/metadata?classKey=DATA_ELEMENT"
  -H "Content-Type:application/csv" -u admin:district
```

The formats for the currently supported object types for CSV import are listed in the following sections.

Data elements

Data Element CSV Format

Index	Column	Required	Value (default first)	Description
1	Name	Yes		Name. Max 230 char. Unique.
2	UID	No	UID	Stable identifier. Exactly 11 alpha-numeric characters, beginning with a character. Will be generated by system if not specified.
3	Code	No		Stable code. Max 50 char.
4	Short name	No	50 first char of name	Will fall back to first 50 characters of name if unspecified. Max 50 char. Unique.
5	Description	No		Free text description.
6	Form name	No		Max 230 char.
7	Domain type	No	AGGREGATE | TRACKER	Domain type for data element, can be aggregate or tracker. Max 16 char.
8	Value type	No	INTEGER | NUMBER | UNIT_INTERVAL | PERCENTAGE | INTEGER_POSITIVE | INTEGER_NEGATIVE | INTEGER_ZERO_OR_POSITIVE | FILE_RESOURCE | COORDINATE |TEXT | LONG_TEXT | LETTER | PHONE_NUMBER | EMAIL | BOOLEAN | TRUE_ONLY | DATE | DATETIME	Value type. Max 16 char.
9	Aggregation type	No	SUM | AVERAGE | AVERAGE_SUM_ORG_UNIT | COUNT | STDDEV | VARIANCE | MIN | MAX | NONE	Aggregation type indicating how to aggregate data in various dimensions. Max 16 char.
10	Category combination	No	UID	UID of category combination. Will default to default category combination if not specified.
11	Url	No		URL to data element resource. Max 255 char.
12	Zero is significant	No	false | true	Indicates whether zero values will be stored for this data element.
13	Option set	No	UID	UID of option set to use for data.
14	Comment option set	No	UID	UID of option set to use for comments.

An example of a CSV file for data elements can be seen below. The first row will always be ignored. Note how you can skip columns and rely on default values to be used by the system. You can also skip columns which you do not use which appear to the right of the ones

```
name,uid,code,shortname,description
"Women participated skill development training",,"D0001","Women participated in training"
"Women participated community organizations",,"D0002","Women participated in organizations"
```

Organisation units

Organisation Unit CSV Format

Index	Column	Required	Value (default first)	Description
1	Name	Yes		Name. Max 230 characters. Unique.
2	UID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.
3	Code	No		Stable code. Max 50 char.
4	Parent	No	UID	UID of parent organisation unit.
5	Short name	No	50 first char of name	Will fall back to first 50 characters of name if unspecified. Max 50 characters. Unique.
6	Description	No		Free text description.
7	Opening date	No	1970-01-01	Opening date of organisation unit in YYYY-MM-DD format.
8	Closed date	No		Closed date of organisation unit in YYYY-MM-DD format, skip if currently open.
9	Comment	No		Free text comment for organisation unit.
10	Feature type	No	NONE | MULTI_POLYGON | POLYGON | POINT | SYMBOL	Geospatial feature type.
11	Coordinates	No		Coordinates used for geospatial analysis in Geo JSON format.
12	URL	No		URL to organisation unit resource. Max 255 char.
13	Contact person	No		Contact person for organisation unit. Max 255 char.
14	Address	No		Address for organisation unit. Max 255 char.
15	Email	No		Email for organisation unit. Max 150 char.
16	Phone number	No		Phone number for organisation unit. Max 150 char.

A minimal example for importing organisation units with a parent unit looks like this:

```
name,uid,code,parent
"West province",,"WESTP","ImspTQPwCqd"
"East province",,"EASTP","ImspTQPwCqd"
```

Validation rules

Validation Rule CSV Format

Index	Column	Required	Value (default first)	Description
1	Name	Yes		Name. Max 230 characters. Unique.
2	UID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.
3	Code	No		Stable code. Max 50
4	Description	No		Free text description.
5	Instruction	No		Free text instruction.
6	Importance	No	MEDIUM | HIGH | LOW	Importance of validation rule.
7	Rule type (ignored)	No	VALIDATION | SURVEILLANCE	Type of validation rule.
8	Operator	No	equal_to | not_equal_to | greater_than | greater_than_or_equal_to | less_than | less_than_or_equal_to | compulsory_pair | exclusive_pair	Expression operator.
9	Period type	No	Monthly | Daily | Weekly | Quarterly | SixMontly | Yearly	Period type.
10	Left side expression	Yes		Mathematical formula based on data element and option combo UIDs.
11	Left side expression description	Yes		Free text.
12	Left side missing value strategy	No	SKIP_IF_ANY_VALUE_MISSING | SKIP_IF_ALL_VALUES_MISSING | NEVER_SKIP	Behavior in case of missing values in left side expression.
13	Right side expression	Yes		Mathematical formula based on data element and option combo UIDs.
14	Right side expression description	Yes		Free text.
15	Right side missing value strategy	No	SKIP_IF_ANY_VALUE_MISSING | SKIP_IF_ALL_VALUES_MISSING | NEVER_SKIP	Behavior in case of missing values in right side expression.

Option sets

Option Set CSV Format

Index	Column	Required	Value (default first)	Description
1	OptionSetName	Yes		Name. Max 230 characters. Unique. Should be repeated for each option.
2	OptionSetUID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified. Should be repeated for each option.
3	OptionSetCode	No		Stable code. Max 50 char. Should be repeated for each option.
4	OptionName	Yes		Option name. Max 230 characters.
5	OptionUID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.
6	OptionCode	Yes		Stable code. Max 50 char.

The format for option sets is special. The three first values represent an option set. The three last values represent an option. The first three values representing the option set should be repeated for each option.

```
optionsetname,optionsetuid,optionsetcode,optionname,optionuid,optioncode
"Color",,"COLOR","Blue",,"BLUE"
"Color",,"COLOR","Green",,"GREEN"
"Color",,"COLOR","Yellow",,"YELLOW"
"Sex",,,"Male",,"MALE"
"Sex",,,"Female",,"FEMALE"
"Sex",,,"Unknown",,"UNKNOWN"
"Result",,,"High",,"HIGH"
"Result",,,"Medium",,"MEDIUM"
"Result",,,"Low",,"LOW"
"Impact","cJ82jd8sd32","IMPACT","Great",,"GREAT"
"Impact","cJ82jd8sd32","IMPACT","Medium",,"MEDIUM"
"Impact","cJ82jd8sd32","IMPACT","Poor",,"POOR"
```

Option group

Option Group CSV Format

Index	Column	Required	Value (default first)	Description
1	OptionGroupName	Yes	Name. Max 230 characters. Unique. Should be repeated for each option.	

Index	Column	Required	Value (default first)	Description
2	OptionGroupUid	No	Stable identifier. Max 11 char. Will be generated by system if not specified. Should be repeated for each option.	
3	OptionGroupCode	No	Stable code. Max 50 char. Should be repeated for each option.	
4	OptionGroupShortName	Yes	Short Name. Max 50 characters. Unique. Should be repeated for each option.	
5	OptionSetUid	Yes	Stable identifier. Max 11 char. Should be repeated for each option.	
6	OptionUid	No	Stable identifier. Max 11 char.	
7	OptionCode	No	Stable code. Max 50 char.	

Sample OptionGroup CSV payload

```
optionGroupName,optionGroupUid,optionGroupCode,optionGroupShortName,optionSetUid,optionUid,optionCode
optionGroupA,,,groupA,xmRubJIhmaK,,OptionA
optionGroupA,,,groupA,xmRubJIhmaK,,OptionB
optionGroupB,,,groupB,QYDAByFgTr1,,OptionC
```

Option Group Set

Option Group Set CSV Format

Index	Column	Required	Value (default first)	Description
1	OptionGroupSetName	Yes	Name. Max 230 characters. Unique. Should be repeated for each option.	
2	OptionGroupSetUid	No	Stable identifier. Max 11 char. Will be generated by system if not specified. Should be repeated for each option.	
3	OptionGroupSetCode	No	Stable code. Max 50 char. Should be repeated for each option.	
4	OptionGroupSetDescription	No	Description. Should be repeated for each option.	
5	DataDimension	No	TRUE, FALSE	
6	OptionSetUid	No	OptionSet UID. Stable identifier. Max 11 char.	

Sample OptionGroupSet CSV payload

```
name,uid,code,description,datadimension,optionsetuid
optiongroupsetA,,,,,xmRubJIhmaK
optiongroupsetB,,,,false,QYDAByFgTr1
```

To add OptionGroups to an imported OptionGroupSet, follow the steps as importing collection membership

Collection membership

In addition to importing objects, you can also choose to only import the group-member relationship between an object and a group. Currently, the following group and object pairs are supported

- Organisation Unit Group - Organisation Unit
- Data Element Group - Data Element
- Indicator Group - Indicator
- Option Group Set - Option Group

The CSV format for these imports are the same

Collection membership CSV Format

Index	Column	Required	Value (default first)	Description
1	UID	Yes	UID	The UID of the collection to add an object to
2	UID	Yes	UID	The UID of the object to add to the collection

Other objects

Data Element Group, Category Option, Category Option Group, Organisation Unit Group CSV Format

Index	Column	Required	Value (default first)	Description
1	Name	Yes		Name. Max 230 characters. Unique.
2	UID	No	UID	Stable identifier. Max 11 chars. Will be generated by system if not specified.
3	Code	No		Stable code. Max 50 char.
4	Short name	No		Short name. Max 50 characters.

An example of category options looks like this:

```
name,uid,code,shortname
"Male",,"MALE"
"Female",,"FEMALE"
```

Deleted objects

The deleted objects resource provides a log of metadata objects being deleted.

```
/api/deletedObjects
```

Whenever an object of type metadata is deleted, a log is being kept of the uid, code, the type and the time of when it was deleted. This API is available at `/api/deletedObjects` field filtering and object filtering works similarly to other metadata resources.

Get deleted objects of type data elements:

```
GET /api/deletedObjects.json?klass=DataElement
```

Get deleted object of type indicator which was deleted in 2015 and forward:

```
GET /api/deletedObjects.json?klass=Indicator&deletedAt=2015-01-01
```

Favorites

Certain types of metadata objects can be marked as favorites for the currently logged in user. This applies currently for dashboards.

```
/api/dashboards/<uid>/favorite
```

To make a dashboard a favorite you can make a *POST* request (no content type required) to a URL like this:

```
/api/dashboards/iMnYyBfSxmM/favorite
```

To remove a dashboard as a favorite you can make a *DELETE* request using the same URL as above.

The favorite status will appear as a boolean *favorite* field on the object (e.g. the dashboard) in the metadata response.

Subscriptions

A logged user can subscribe to certain types of objects. Currently subscribable objects are those of type Chart, EventChart, EventReport, Map, ReportTable and Visualization.

> **Note**
>
> The Chart and ReportTable objects are deprecated. Use Visualization instead.

To get the subscribers of an object (return an array of user IDs) you can make a *GET* request:

```
/api/<object-type>/<object-id>/subscribers
```

See example as follows:

```
/api/charts/DkPKc1EUmC2/subscribers
```

To check whether the current user is subscribed to an object (returns a boolean) you can perform a *GET* call:

```
/api/<object-type>/<object-id>/subscribed
```

See example as follows:

```
/api/charts/DkPKc1EUmC2/subscribed
```

To subscribe/de-subscribe to an object you perform a *POST/DELETE* request (no content type required):

```
/api/<object-type>/<object-id>/subscriber
```

File resources

File resources are objects used to represent and store binary content. The *FileResource* object itself contains the file meta-data (name, Content-Type, size, etc.) as well as a key allowing retrieval of the contents from a database-external file store. The *FileResource* object is stored in the database like any other but the content (file) is stored elsewhere and is retrievable using the contained reference *(storageKey)*.

```
/api/fileResources
```

The contents of file resources are not directly accessible but are referenced from other objects (such as data values) to store binary content of virtually unlimited size.

Creation of the file resource itself is done through the `/api/fileResources` endpoint as a multipart upload POST-request:

```
curl "https://server/api/fileResources" -X POST
  -F "file=@/Path/to/file;filename=name-of-file.png"
```

The only form parameter required is the *file* which is the file to upload. The filename and content-type should also be included in the request but will be replaced with defaults when not supplied.

On successfully creating a file resource the returned data will contain a `response` field which in turn contains the `fileResource` like this:

```
{
  "httpStatus": "Accepted",
  "httpStatusCode": 202,
  "status": "OK",
  "response": {
    "responseType": "FileResource",
    "fileResource": {
      "name": "name-of-file.png",
      "created": "2015-10-16T16:34:20.654+0000",
      "lastUpdated": "2015-10-16T16:34:20.667+0000",
      "externalAccess": false,
      "publicAccess": "--------",
      "user": { ... },
      "displayName": "name-of-file.png",
      "contentType": "image/png",
      "contentLength": 512571,
      "contentMd5": "4e1fc1c3f999e5aa3228d531e4adde58",
      "storageStatus": "PENDING",
      "id": "xm4JwRwke0i"
```

```
    }
  }
}
```

Note that the response is a *202 Accepted*, indicating that the returned resource has been submitted for background processing (persisting to the external file store in this case). Also, note the `storageStatus` field which indicates whether the contents have been stored or not. At this point, the persistence to the external store is not yet finished (it is likely being uploaded to a cloud-based store somewhere) as seen by the `PENDING` status.

Even though the content has not been fully stored yet the file resource can now be used, for example as referenced content in a data value (see Working with file data values). If we need to check the updated *storageStatus* or otherwise retrieve the metadata of the file, the `fileResources` endpoint can be queried.

```
curl "https://server/api/fileResources/xm4JwRwke0i" -H "Accept: application/json"
```

This request will return the `FileResource` object as seen in the response of the above example.

File resource constraints

- File resources *must* be referenced (assigned) from another object in order to be persisted in the long term. A file resource which is created but not referenced by another object such as a data value is considered to be in *staging*. Any file resources which are in this state and are older than *two hours* will be marked for deletion and will eventually be purged from the system.
- The ID returned by the initial creation of the file resource is not retrievable from any other location unless the file resource has been referenced (in which the ID will be stored as the reference), so losing it will require the POST request to be repeated and a new object to be created. The *orphaned* file resource will be cleaned up automatically.
- File resource objects are *immutable*, meaning modification is not allowed and requires creating a completely new resource instead.

File resource blocklist

Certain types of files are blocked from being uploaded for security reasons.

The following content types are blocked.

Content type	Content type
text/html	application/x-ms-dos-executable
text/css	application/vnd.microsoft.portable-executable
text/javascript	application/vnd.apple.installer+xml
font/otf	application/vnd.mozilla.xul+xml
application/x-shockwave-flash	application/x-httpd-php
application/vnd.debian.binary-package	application/x-sh
application/x-rpm	application/x-csh
application/java-archive	

The following file extensions are blocked.

File extension	File extension	File extension
html	deb	xul
htm	rpm	php
css	jar	bin
js	jsp	sh
mjs	exe	csh
otf	msi	bat
swf	mpkg	

Metadata versioning

This section explains the Metadata Versioning APIs available starting 2.24

- `/api/metadata/version`: This endpoint will return the current metadata version of the system on which it is invoked.

Query Parameters

Name	Required	Description
versionName	false	If this parameter is not specified, it will return the current version of the system or otherwise it will return the details of the versionName passed as parameter. (versionName is of the syntax "Version_<id>"

Get metadata version examples

Example: Get the current metadata version of this system

Request:

```
/api/metadata/version
```

Response:

```
{
    "name": "Version_4",
    "created": "2016-06-30T06:01:28.684+0000",
    "lastUpdated": "2016-06-30T06:01:28.685+0000",
    "externalAccess": false,
    "displayName": "Version_4",
    "type": "BEST_EFFORT",
    "hashCode": "848bf6edbaf4faeb7d1a1169445357b0",
    "id": "Ayz2AEMB6ry"
}
```

Example: Get the details of version with name "Version_2"

Request:

```
/api/metadata/version?versionName=Version_2
```

Response:

```
{
    "name": "Version_2",
    "created": "2016-06-30T05:59:33.238+0000",
    "lastUpdated": "2016-06-30T05:59:33.239+0000",
    "externalAccess": false,
    "displayName": "Version_2",
    "type": "BEST_EFFORT",
    "hashCode": "8050fb1a604e29d5566675c86d02d10b",
    "id": "SaNyhusVxBG"
}
```

- `/api/metadata/version/history`: This endpoint will return the list of all metadata versions of the system on which it is invoked.

Query Parameters

Name	Required	Description
baseline	false	If this parameter is not specified, it will return list of all metadata versions. Otherwise we need to pass a versionName parameter of the form "Version_<id>". It will then return the list of versions present in the system which were created after the version name supplied as the query parameter.

Get the list of all metadata versions

Example: Get the list of all versions in this system

Request:

```
/api/metadata/version/history
```

Response:

```
{
    "metadataversions": [
        {
            "name": "Version_1",
            "type": "BEST_EFFORT",
            "created": "2016-06-30T05:54:41.139+0000",
            "id": "SjnhUp6r4hG",
            "hashCode": "fd1398ff7ec9fcfd5b59d523c8680798"
        },
        {
            "name": "Version_2",
            "type": "BEST_EFFORT",
            "created": "2016-06-30T05:59:33.238+0000",
            "id": "SaNyhusVxBG",
            "hashCode": "8050fb1a604e29d5566675c86d02d10b"
        },
        {
```

```
            "name": "Version_3",
            "type": "BEST_EFFORT",
            "created": "2016-06-30T06:01:23.680+0000",
            "id": "FVkGzSjAAYg",
            "hashCode": "70b779ea448b0da23d8ae0bd59af6333"
        }
    ]
}
```


Example: Get the list of all versions in this system created after "Version_2"

Request:

```
/api/metadata/version/history?baseline=Version_2
```

Response:


```
{
    "metadataversions": [
        {
            "name": "Version_3",
            "type": "BEST_EFFORT",
            "created": "2016-06-30T06:01:23.680+0000",
            "id": "FVkGzSjAAYg",
            "hashCode": "70b779ea448b0da23d8ae0bd59af6333"
        },
        {
            "name": "Version_4",
            "type": "BEST_EFFORT",
            "created": "2016-06-30T06:01:28.684+0000",
            "id": "Ayz2AEMB6ry",
            "hashCode": "848bf6edbaf4faeb7d1a1169445357b0"
        }
    ]
}
```


- `/api/metadata/version/create`: This endpoint will create the metadata version for the version type as specified in the parameter.

Query Parameters

Name	Required	Description
type	true	The type of metadata version which needs to be created. • BEST_EFFORT • ATOMIC

Users can select the type of metadata which needs to be created. Metadata Version type governs how the importer should treat the given version. This type will be used while importing the metadata. There are two types of metadata.

- *BEST_EFFORT*: This type suggests that missing references can be ignored and the importer can continue importing the metadata (e.g. missing data elements on a data element group import).
- *ATOMIC*: This type ensures a strict type checking of the metadata references and the metadata import will fail if any of the references do not exist.

> **Note**
>
> It's recommended to have an ATOMIC type of versions to ensure that all systems (central and local) have the same metadata. Any missing reference is caught in the validation phase itself. Please see the importer details for a full explanation.

Create metadata version

Example: Create metadata version of type BEST_EFFORT

Request:

```
curl -X POST -u admin:district "https://play.dhis2.org/dev/api/metadata/version/create?
type=BEST_EFFORT"
```

Response:

```
{
    "name": "Version_1",
    "created": "2016-06-30T05:54:41.139+0000",
    "lastUpdated": "2016-06-30T05:54:41.333+0000",
    "externalAccess": false,
    "publicAccess": "--------",
    "user": {
        "name": "John Traore",
        "created": "2013-04-18T17:15:08.407+0000",
        "lastUpdated": "2016-04-06T00:06:06.571+0000",
        "externalAccess": false,
        "displayName": "John Traore",
        "id": "xE7jOejl9FI"
    },
    "displayName": "Version_1",
    "type": "BEST_EFFORT",
    "hashCode": "fd1398ff7ec9fcfd5b59d523c8680798",
    "id": "SjnhUp6r4hG"
}
```

- `/api/metadata/version/{versionName}/data`: This endpoint will download the actual metadata specific to the version name passed as path parameter.
- `/api/metadata/version/{versionName}/data.gz`: This endpoint will download the actual metadata specific to the version name passed as path parameter in a compressed format (gzipped).

Path parameters

Name	Required	Description
versionName	true	Path parameter of the form "Version_<id>" so that the API downloads the specific version

Download version metadata

Example: Get the actual metadata for "Version 5"

Request:

```
curl -u admin:district "https://play.dhis2.org/dev/api/metadata/version/Version_5/data"
```

Response:

```
{
    "date": "2016-06-30T06:10:23.120+0000",
    "dataElements": [
        {
            "code": "ANC 5th Visit",
            "created": "2016-06-30T06:10:09.870+0000",
            "lastUpdated": "2016-06-30T06:10:09.870+0000",
            "name": "ANC 5th Visit",
            "id": "sCuZKDsix7Y",
            "shortName": "ANC 5th Visit ",
            "aggregationType": "SUM",
            "domainType": "AGGREGATE",
            "zeroIsSignificant": false,
            "valueType": "NUMBER",
            "categoryCombo": {
                "id": "p0KPaWEg3cf"
            },
            "user": {
                "id": "xE7jOejl9FI"
            }
        }
    ]
}
```

Metadata Synchronization

This section explains the Metadata Synchronization API available starting 2.24

- `/api/metadata/sync`: This endpoint performs metadata sync of the version name passed in the query parameter by downloading and importing the specified version from the remote server as defined in the settings app.

Query parameters

Name	Required	Description
versionName	true	versionName query parameter of the form "Version_<id>" . The api downloads this version from the remote server and imports it in the local system.

- This API should be used with utmost care. Please note that there is an alternate way to achieve sync in a completely automated manner by leveraging the Metadata Sync Task from the "Data Administration" app. See Chapter 22, Section 22.17 of User Manual for more details regarding Metadata Sync Task.
- This sync API can alternatively be used to sync metadata for the versions which have failed from the metadata sync scheduler. Due to its dependence on the given metadata version number, care should be taken for the order in which this gets invoked. E.g. If this api is used to sync some higher version from the central instance, then the sync might fail as the metadata dependencies are not present in the local instance.
- Assume the local instance is at `Version_12` and if this endpoint is used to sync `Version_15` (of type `BEST_EFFORT`) from the central instance, the scheduler will start syncing metadata from `Version_16`. So the local instance will not have the metadata versions between `Version_12` and `Version_15`. You need to manually sync the missing versions using these endpoints only.

Sync metadata version

Example: Sync Version_6 from central system to this system

Request:

```
curl -u admin:district "https://play.dhis2.org/dev/api/metadata/sync?versionName=Version_6"
```

Data values

This section is about sending and reading data values.

```
/api/33/dataValueSets
```

Sending data values

A common use-case for system integration is the need to send a set of data values from a third-party system into DHIS. In this example, we will use the DHIS2 demo on `http://play.dhis2.org/demo` as basis. We assume that we have collected case-based data using a simple software client running on mobile phones for the *Mortality <5 years* data set in the community of *Ngelehun CHC* (in *Badjia* chiefdom, *Bo* district) for the month of January 2014. We have now aggregated our data into a statistical report and want to send that data to the DHIS2 instance. The base URL to the demo API is `http://play.dhis2.org/demo/api`. The following links are relative to the base URL.

The resource which is most appropriate for our purpose of sending data values is the `/api/dataValueSets` resource. A data value set represents a set of data values which have a relationship, usually from being captured off the same data entry form. The format looks like this:

```
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0" dataSet="dataSetID"
  completeDate="date" period="period" orgUnit="orgUnitID" attributeOptionCombo="aocID">
  <dataValue dataElement="dataElementID"
    categoryOptionCombo="cocID" value="1" comment="comment1"/>
  <dataValue dataElement="dataElementID"
    categoryOptionCombo="cocID" value="2" comment="comment2"/>
  <dataValue dataElement="dataElementID"
    categoryOptionCombo="cocID" value="3" comment="comment3"/>
</dataValueSet>
```

JSON is supported in this format:

```
{
    "dataSet": "dataSetID",
    "completeDate": "date",
    "period": "period",
    "orgUnit": "orgUnitID",
    "attributeOptionCombo": "aocID",
    "dataValues": [
        {
            "dataElement": "dataElementID",
            "categoryOptionCombo": "cocID",
            "value": "1",
            "comment": "comment1"
        },
        {
            "dataElement": "dataElementID",
            "categoryOptionCombo": "cocID",
            "value": "2",
            "comment": "comment2"
        },
        {
            "dataElement": "dataElementID",
            "categoryOptionCombo": "cocID",
            "value": "3",
            "comment": "comment3"
        }
    ]
}
```

CSV is supported in this format:

```
"dataelement","period","orgunit","catoptcombo","attroptcombo","value","strby","lstupd","cmt"
"dataElementID","period","orgUnitID","cocID","aocID","1","username","2015-04-01","comment1"
"dataElementID","period","orgUnitID","cocID","aocID","2","username","2015-04-01","comment2"
"dataElementID","period","orgUnitID","cocID","aocID","3","username","2015-04-01","comment3"
```

> **Note**
>
> Please refer to the date and period section above for time formats.

From the example, we can see that we need to identify the period, the data set, the org unit (facility) and the data elements for which to report.

To obtain the identifier for the data set we make a request to the `/api/dataSets` resource. From there we find and follow the link to the *Mortality < 5 years* data set which leads us to `/api/dataSets/pBOMPrpg1QX`. The resource representation for the *Mortality < 5 years* data set conveniently advertises links to the data elements which are members of it. From here we can follow these links and obtain the identifiers of the data elements. For brevity we will only report on three data elements: *Measles* with id `f7n9E0hX8qk`, *Dysentery* with id `Ix2HsbDMLea` and *Cholera* with id `eY5ehpbEsB7`.

What remains is to get hold of the identifier of the organisation unit. The *dataSet* representation conveniently provides a link to organisation units which report on it so we search for *Ngelehun CHC* and follow the link to the HTML representation at `/api/organisationUnits/DiszpKrYNg8`, which tells us that the identifier of this org unit is `DiszpKrYNg8`.

From our case-based data, we assume that we have 12 cases of measles, 14 cases of dysentery and 16 cases of cholera. We have now gathered enough information to be able to put together the XML data value set message:

```
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0" dataSet="pBOMPrpg1QX"
  completeDate="2014-02-03" period="201401" orgUnit="DiszpKrYNg8">
  <dataValue dataElement="f7n9E0hX8qk" value="12"/>
  <dataValue dataElement="Ix2HsbDMLea" value="14"/>
  <dataValue dataElement="eY5ehpbEsB7" value="16"/>
</dataValueSet>
```

In JSON format:

```
{
    "dataSet": "pBOMPrpg1QX",
    "completeDate": "2014-02-03",
    "period": "201401",
    "orgUnit": "DiszpKrYNg8",
    "dataValues": [
        {
            "dataElement": "f7n9E0hX8qk",
            "value": "1"
        },
        {
            "dataElement": "Ix2HsbDMLea",
            "value": "2"
        },
        {
            "dataElement": "eY5ehpbEsB7",
            "value": "3"
        }
    ]
}
```

To perform functional testing we will use the *curl* tool which provides an easy way of transferring data using HTTP. First, we save the data value set XML content in a file called `datavalueset.xml`. From the directory where this file resides we invoke the following from the command line:

```
curl -d @datavalueset.xml "https://play.dhis2.org/demo/api/33/dataValueSets"
  -H "Content-Type:application/xml" -u admin:district
```

For sending JSON content you must set the content-type header accordingly:

```
curl -d @datavalueset.json "https://play.dhis2.org/demo/api/33/dataValueSets"
  -H "Content-Type:application/json" -u admin:district
```

The command will dispatch a request to the demo Web API, set `application/xml` as the content-type and authenticate using `admin/district` as username/password. If all goes well this will return a `200 OK` HTTP status code. You can verify that the data has been received by opening the data entry module in DHIS2 and select the org unit, data set and period used in this example.

The API follows normal semantics for error handling and HTTP status codes. If you supply an invalid username or password, `401 Unauthorized` is returned. If you supply a content-type other than `application/xml`, `415 Unsupported Media Type` is returned. If the XML content is invalid according to the DXF namespace, `400 Bad Request` is returned. If you provide an invalid identifier in the XML content, `409 Conflict` is returned together with a descriptive message.

Sending bulks of data values

The previous example showed us how to send a set of related data values sharing the same period and organisation unit. This example will show us how to send large bulks of data values which don't necessarily are logically related.

Again we will interact with the `/api/dataValueSets` resource. This time we will not specify the `dataSet` and `completeDate` attributes. Also, we will specify the `period` and `orgUnit` attributes on the individual data value elements instead of on the outer data value set element. This will enable us to send data values for various periods and organisation units:

```
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0">
  <dataValue dataElement="f7n9E0hX8qk"
    period="201401" orgUnit="DiszpKrYNg8" value="12"/>
  <dataValue dataElement="f7n9E0hX8qk"
    period="201401" orgUnit="FNnj3jKGS7i" value="14"/>
  <dataValue dataElement="f7n9E0hX8qk"
    period="201402" orgUnit="DiszpKrYNg8" value="16"/>
  <dataValue dataElement="f7n9E0hX8qk"
    period="201402" orgUnit="Jkhdsf8sdf4" value="18"/>
</dataValueSet>
```

In JSON format:

```
{
    "dataValues": [
        {
            "dataElement": "f7n9E0hX8qk",
            "period": "201401",
            "orgUnit": "DiszpKrYNg8",
            "value": "12"
        },
        {
            "dataElement": "f7n9E0hX8qk",
            "period": "201401",
```

```
            "orgUnit": "FNnj3jKGS7i",
            "value": "14"
        },
        {
            "dataElement": "f7n9E0hX8qk",
            "period": "201402",
            "orgUnit": "DiszpKrYNg8",
            "value": "16"
        },
        {
            "dataElement": "f7n9E0hX8qk",
            "period": "201402",
            "orgUnit": "Jkhdsf8sdf4",
            "value": "18"
        }
    ]
}
```

In CSV format:

```
"dataelement","period","orgunit","categoryoptioncombo","attributeoptioncombo","value"
"f7n9E0hX8qk","201401","DiszpKrYNg8","bRowv6yZOF2","bRowv6yZOF2","1"
"Ix2HsbDMLea","201401","DiszpKrYNg8","bRowv6yZOF2","bRowv6yZOF2","2"
"eY5ehpbEsB7","201401","DiszpKrYNg8","bRowv6yZOF2","bRowv6yZOF2","3"
```

We test by using curl to send the data values in XML format:

```
curl -d @datavalueset.xml "https://play.dhis2.org/demo/api/33/dataValueSets"
  -H "Content-Type:application/xml" -u admin:district
```

Note that when using CSV format you must use the binary data option to preserve the line-breaks in the CSV file:

```
curl --data-binary @datavalueset.csv "https://play.dhis2.org/demo/24/api/dataValueSets"
  -H "Content-Type:application/csv" -u admin:district
```

The data value set resource provides an XML response which is useful when you want to verify the impact your request had. The first time we send the data value set request above the server will respond with the following import summary:

```
<importSummary>
  <dataValueCount imported="2" updated="1" ignored="1"/>
  <dataSetComplete>false</dataSetComplete>
</importSummary>
```

This message tells us that 3 data values were imported, 1 data value was updated while zero data values were ignored. The single update comes as a result of us sending that data value in the previous example. A data value will be ignored if it references a non-existing data element, period, org unit or data set. In our case, this single ignored value was caused by the last data value having an invalid reference to org unit. The data set complete element will display the date of which the data value set was completed, or false if no data element attribute was supplied.

Import parameters

The import process can be customized using a set of import parameters:

Import parameters

Parameter	Values (default first)	Description
dataElementIdScheme	id | name | code | attribute:ID	Property of the data element object to use to map the data values.
orgUnitIdScheme	id | name | code | attribute:ID	Property of the org unit object to use to map the data values.
categoryOptionComboIdScheme	id | name | code | attribute:ID	Property of the category option combo and attribute option combo objects to use to map the data values.
idScheme	id | name | code| attribute:ID	Property of all objects including data elements, org units and category option combos, to use to map the data values.
preheatCache	false | true	Indicates whether to preload metadata caches before starting to import data values, will speed up large import payloads with high metadata cardinality.
dryRun	false | true	Whether to save changes on the server or just return the import summary.
importStrategy	CREATE | UPDATE | CREATE_AND_UPDATE | DELETE	Save objects of all, new or update import status on the server.
skipExistingCheck	false | true	Skip checks for existing data values. Improves performance. Only use for empty databases or when the data values to import do not exist already.
skipAudit	false | true	Skip audit, meaning audit values will not be generated. Improves performance at the cost of ability to audit changes. Requires authority "F_SKIP_DATA_IMPORT_AUDIT".

Parameter	Values (default first)	Description
async	false | true	Indicates whether the import should be done asynchronous or synchronous. The former is suitable for very large imports as it ensures that the request does not time out, although it has a significant performance overhead. The latter is faster but requires the connection to persist until the process is finished.
force	false | true	Indicates whether the import should be forced. Data import could be rejected for various reasons of data set locking for example due to approval, data input period, expiry days, etc. In order to override such locks and force data input one can use data import with force=true. However, one needs to be a *superuser* for this parameter to work.

All parameters are optional and can be supplied as query parameters in the request URL like this:

```
/api/33/dataValueSets?dataElementIdScheme=code&orgUnitIdScheme=name
  &dryRun=true&importStrategy=CREATE
```

They can also be supplied as XML attributes on the data value set element like below. XML attributes will override query string parameters.

```
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0" dataElementIdScheme="code"
  orgUnitIdScheme="name" dryRun="true" importStrategy="CREATE">
</dataValueSet>
```

Note that the `preheatCache` parameter can have a huge impact on performance. For small import files, leaving it to false will be fast. For large import files which contain a large number of distinct data elements and organisation units, setting it to true will be orders of magnitude faster.

Data value requirements

Data value import supports a set of value types. For each value type, there is a special requirement. The following table lists the edge cases for value types.

Value type requirements

Value type	Requirements	Comment
BOOLEAN	true | True | TRUE | false | False | FALSE | 1 | 0 | t | f |	Used when the value is a boolean, true or false value. The import service does not care if the input begins with an uppercase or lowercase letter, or if it's all uppercase.

Identifier schemes

Regarding the id schemes, by default the identifiers used in the XML messages use the DHIS2 stable object identifiers referred to as `UID`. In certain interoperability situations we might experience that an external system decides the identifiers of the objects. In that case we can use the `code` property of the organisation units and other objects to set fixed identifiers. When importing data values we hence need to reference the code property instead of the identifier property of these metadata objects. Identifier schemes can be specified in the XML message as well as in the request as query parameters. To specify it in the XML payload you can do this:

```
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0"
  dataElementIdScheme="CODE" orgUnitIdScheme="UID" idScheme="CODE">
</dataValueSet>
```

The parameter table above explains how the id schemes can be specified as query parameters. The following rules apply for what takes precedence:

- Id schemes defined in the XML or JSON payload take precedence over id schemes defined as URL query parameters.
- Specific id schemes including dataElementIdScheme and orgUnitIdScheme take precedence over the general idScheme.
- The default id scheme is UID, which will be used if no explicit id scheme is defined.

The following identifier schemes are available.

- uid (default)
- code
- name
- attribute (followed by UID of attribute)

The attribute option is special and refers to meta-data attributes which have been marked as *unique*. When using this option, `attribute` must be immediately followed by the identifier of the attribute, e.g. "attribute:DnrLSdo4hMl".

Async data value import

Data values can be sent and imported in an asynchronous fashion by supplying an `async` query parameter set to *true*:

```
/api/33/dataValueSets?async=true
```

This will initiate an asynchronous import job for which you can monitor the status at the task summaries API. The API response indicates the unique identifier of the job, type of job and the URL you can use to monitor the import job status. The response will look similar to this:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Initiated dataValueImport",
    "response": {
        "name": "dataValueImport",
        "id": "YR1UxOUXmzT",
        "created": "2018-08-20T14:17:28.429",
        "jobType": "DATAVALUE_IMPORT",
        "relativeNotifierEndpoint": "/api/system/tasks/DATAVALUE_IMPORT/YR1UxOUXmzT"
    }
}
```

Please read the section on *asynchronous task status* for more information.

CSV data value format

The following section describes the CSV format used in DHIS2. The first row is assumed to be a header row and will be ignored during import.

CSV format of DHIS2

Column	Required	Description
Data element	Yes	Refers to ID by default, can also be name and code based on selected id scheme
Period	Yes	In ISO format
Org unit	Yes	Refers to ID by default, can also be name and code based on selected id scheme
Category option combo	No	Refers to ID
Attribute option combo	No	Refers to ID (from version 2.16)
Value	No	Data value
Stored by	No	Refers to username of user who entered the value
Last updated	No	Date in ISO format
Comment	No	Free text comment
Follow up	No	true or false

An example of a CSV file which can be imported into DHIS2 is seen below.

```
"dataelement","period","orgunit","catoptcombo","attroptcombo","value","storedby","timestamp"
"DUSpd8Jq3M7","201202","gP6hn503KUX","Prlt0C1RF0s",,"7","bombali","2010-04-17"
"DUSpd8Jq3M7","201202","gP6hn503KUX","V6L425pT3A0",,"10","bombali","2010-04-17"
"DUSpd8Jq3M7","201202","OjTS752GbZE","V6L425pT3A0",,"9","bombali","2010-04-06"
```

Generating data value set template

To generate a data value set template for a certain data set you can use the `/api/dataSets/<id>/dataValueSet` resource. XML and JSON response formats are supported. Example:

```
/api/dataSets/BfMAe6Itzgt/dataValueSet.json
```

The parameters you can use to further adjust the output are described below:

Data values query parameters

Query parameter	Required	Description
period	No	Period to use, will be included without any checks.
orgUnit	No	Organisation unit to use, supports multiple orgUnits, both id and code can be used.
comment	No	Should comments be include, default: Yes.
orgUnitIdScheme	No	Organisation unit scheme to use, supports id | code.
dataElementIdScheme	No	Data-element scheme to use, supports id | code.

Reading data values

This section explains how to retrieve data values from the Web API by interacting with the *dataValueSets* resource. Data values can be retrieved in *XML*, *JSON* and *CSV* format. Since we want to read data we will use the *GET* HTTP verb. We will also specify that we are interested in the XML resource representation by including an `Accept` HTTP header with our request. The following query parameters are required:

Data value set query parameters

Parameter	Description
dataSet	Data set identifier. Can be repeated any number of times.
dataElementGroup	Data element group identifier. Can be repeated any number of times.
period	Period identifier in ISO format. Can be repeated any number of times.
startDate	Start date for the time span of the values to export.
endDate	End date for the time span of the values to export.
orgUnit	Organisation unit identifier. Can be repeated any number of times.
children	Whether to include the children in the hierarchy of the organisation units.
orgUnitGroup	Organisation unit group identifier. Can be repeated any number of times.
attributeOptionCombo	Attribute option combo identifier. Can be repeated any number of times.
includeDeleted	Whether to include deleted data values.
lastUpdated	Include only data values which are updated since the given time stamp.
lastUpdatedDuration	Include only data values which are updated within the given duration. The format is <value><time-unit>, where the supported time units are "d" (days), "h" (hours), "m" (minutes) and "s" (seconds).
limit	The max number of results in the response.

Parameter	Description
idScheme	Property of meta data objects to use for data values in response.
dataElementIdScheme	Property of the data element object to use for data values in response.
orgUnitIdScheme	Property of the org unit object to use for data values in response.
categoryOptionComboIdScheme	Property of the category option combo and attribute option combo objects to use for data values in response.
dataSetIdScheme	Property of the data set object to use in the response.

The following response formats are supported:

- xml (application/xml)
- json (application/json)
- csv (application/csv)
- adx (application/adx+xml)

Assuming that we have posted data values to DHIS2 according to the previous section called *Sending data values* we can now put together our request for a single data value set and request it using cURL:

```
curl "https://play.dhis2.org/demo/api/33/dataValueSets?
dataSet=pBOMPrpg1QX&period=201401&orgUnit=DiszpKrYNg8"
  -H "Accept:application/xml" -u admin:district
```

We can also use the start and end dates query parameters to request a larger bulk of data values. I.e. you can also request data values for multiple data sets and org units and a time span in order to export larger chunks of data. Note that the period query parameter takes precedence over the start and end date parameters. An example looks like this:

```
curl "https://play.dhis2.org/demo/api/33/dataValueSets?dataSet=pBOMPrpg1QX&dataSet=BfMAe6Itzgt
  &startDate=2013-01-01&endDate=2013-01-31&orgUnit=YuQRtpLP10I&orgUnit=vWbkYPRmKyS&children=true"
  -H "Accept:application/xml" -u admin:district
```

To retrieve data values which have been created or updated within the last 10 days you can make a request like this:

```
/api/dataValueSets?dataSet=pBOMPrpg1QX&orgUnit=DiszpKrYNg8&lastUpdatedDuration=10d
```

The response will look like this:

```
<?xml version='1.0' encoding='UTF-8'?>
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0" dataSet="pBOMPrpg1QX"
  completeDate="2014-01-02" period="201401" orgUnit="DiszpKrYNg8">
<dataValue dataElement="eY5ehpbEsB7" period="201401" orgUnit="DiszpKrYNg8"
  categoryOptionCombo="bRowv6yZOF2" value="10003"/>
<dataValue dataElement="Ix2HsbDMLea" period="201401" orgUnit="DiszpKrYNg8"
  categoryOptionCombo="bRowv6yZOF2" value="10002"/>
<dataValue dataElement="f7n9E0hX8qk" period="201401" orgUnit="DiszpKrYNg8"
  categoryOptionCombo="bRowv6yZOF2" value="10001"/>
</dataValueSet>
```

You can request the data in JSON format like this:

```
/api/dataValueSets.json?dataSet=pBOMPrpg1QX&period=201401&orgUnit=DiszpKrYNg8
```

The response will look something like this:

```
{
    "dataSet": "pBOMPrpg1QX",
    "completeDate": "2014-02-03",
    "period": "201401",
    "orgUnit": "DiszpKrYNg8",
    "dataValues": [
        {
            "dataElement": "eY5ehpbEsB7",
            "categoryOptionCombo": "bRowv6yZOF2",
            "period": "201401",
            "orgUnit": "DiszpKrYNg8",
            "value": "10003"
        },
        {
            "dataElement": "Ix2HsbDMLea",
            "categoryOptionCombo": "bRowv6yZOF2",
            "period": "201401",
            "orgUnit": "DiszpKrYNg8",
            "value": "10002"
        },
        {
            "dataElement": "f7n9E0hX8qk",
            "categoryOptionCombo": "bRowv6yZOF2",
            "period": "201401",
            "orgUnit": "DiszpKrYNg8",
            "value": "10001"
        }
    ]
}
```

Note that data values are softly deleted, i.e. a deleted value has the `deleted` property set to true instead of being permanently deleted. This is useful when integrating multiple systems in order to communicate deletions. You can include deleted values in the response like this:

```
/api/33/dataValueSets.json?dataSet=pBOMPrpg1QX&period=201401
  &orgUnit=DiszpKrYNg8&includeDeleted=true
```

You can also request data in CSV format like this:

```
/api/33/dataValueSets.csv?dataSet=pBOMPrpg1QX&period=201401
  &orgUnit=DiszpKrYNg8
```

The response will look like this:

```
dataelement,period,orgunit,catoptcombo,attroptcombo,value,storedby,lastupdated,comment,flwup
f7n9E0hX8qk,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,12,system,
2015-04-05T19:58:12.000,comment1,false
Ix2HsbDMLea,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,14,system,
2015-04-05T19:58:12.000,comment2,false
eY5ehpbEsB7,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,16,system,
2015-04-05T19:58:12.000,comment3,false
FTRrcoaog83,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,12,system,
2014-03-02T21:45:05.519,comment4,false
```

The following constraints apply to the data value sets resource:

- At least one data set must be specified.
- Either at least one period or a start date and end date must be specified.
- At least one organisation unit must be specified.
- Organisation units must be within the hierarchy of the organisation units of the authenticated user.
- Limit cannot be less than zero.

Sending, reading and deleting individual data values

This example will show how to send individual data values to be saved in a request. This can be achieved by sending a *POST* request to the `dataValues` resource:

```
/api/dataValues
```

The following query parameters are supported for this resource:

Data values query parameters

Query parameter	Required	Description
de	Yes	Data element identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
co	No	Category option combo identifier, default will be used if omitted
cc	No (must be combined with cp)	Attribute category combo identifier
cp	No (must be combined with cc)	Attribute category option identifiers, separated with ; for multiple values

Query parameter	Required	Description
ds	No	Data set, to check if POST or DELETE is allowed for period and organisation unit. If specified, the data element must be assigned to this data set. If not specified, a data set containing the data element will be chosen to check if the operation is allowed.
value	No	Data value. For boolean values, the following will be accepted: true | True | TRUE | false | False | FALSE | 1 | 0 | t | f |
comment	No	Data comment
followUp	No	Follow up on data value, will toggle the current boolean value

If any of the identifiers given are invalid, if the data value or comment is invalid or if the data is locked, the response will contain the *409 Conflict* status code and descriptive text message. If the operation leads to a saved or updated value, *200 OK* will be returned. An example of a request looks like this:

```
curl "https://play.dhis2.org/demo/api/33/dataValues?de=s46m5MS0hxu
  &pe=201301&ou=DiszpKrYNg8&co=Prlt0C1RF0s&value=12"
  -X POST -u admin:district
```

This resource also allows a special syntax for associating the value to an attribute option combination. This can be done by sending the identifier of the attribute category combination, together with the identifiers of the attribute category options which the value represents within the combination. The category combination is specified with the `cc` parameter, while the category options are specified as a semi-colon separated string with the `cp` parameter. It is necessary to ensure that the category options are all part of the category combination. An example looks like this:

```
curl "https://play.dhis2.org/demo/api/33/dataValues?de=s46m5MS0hxu&ou=DiszpKrYNg8
  &pe=201308&cc=dzjKKQq0cSO&cp=wbrDrL2aYEc;btOyqprQ9e8&value=26"
  -X POST -u admin:district
```

You can retrieve a data value with a request using the *GET* method. The value, comment and followUp params are not applicable in this regard:

```
curl "https://play.dhis2.org/demo/api/33/dataValues?de=s46m5MS0hxu
  &pe=201301&ou=DiszpKrYNg8&co=Prlt0C1RF0s"
  -u admin:district
```

You can delete a data value with a request using the *DELETE* method.

Working with file data values

When dealing with data values which have a data element of type *file* there is some deviation from the method described above. These data values are special in that the contents of the value is a UID reference to a *FileResource* object instead of a self-contained constant. These data values will behave just like other data values which store text content, but should be handled differently in order to produce meaningful input and output.

The process of storing one of these data values roughly goes like this:

1. Upload the file to the `/api/fileResources` endpoint as described in the file resource section.
2. Retrieve the `id` property of the returned *FileResource*.
3. Store the retrieved id *as the value* to the data value using any of the methods described above.

Only one-to-one relationships between data values and file resources are allowed. This is enforced internally so that saving a file resource id in several data values is not allowed and will return an error. Deleting the data value will delete the referenced file resource. Direct deletion of file resources are not possible.

The data value can now be retrieved as any other but the returned data will be the UID of the file resource. In order to retrieve the actual contents (meaning the file which is stored in the file resource mapped to the data value) a GET request must be made to `/api/dataValues/files` mirroring the query parameters as they would be for the data value itself. The `/api/dataValues/files` endpoint only supports GET requests.

It is worth noting that due to the underlying storage mechanism working asynchronously the file content might not be immediately ready for download from the `/api/dataValues/files` endpoint. This is especially true for large files which might require time consuming uploads happening in the background to an external file store (depending on the system configuration). Retrieving the file resource meta-data from the `/api/fileResources/<id>` endpoint allows checking the `storageStatus` of the content before attempting to download it.

ADX data format

From version 2.20 we have included support for an international standard for aggregate data exchange called ADX. ADX is developed and maintained by the Quality Research and Public Health committee of the IHE (Integrating the HealthCare Enterprise). The wiki page detailing QRPH activity can be found at wiki.ihe.net. ADX is still under active development and has now been published for trial implementation. Note that what is implemented currently in DHIS2 is the functionality to read and write adx formatted data, i.e. what is described as Content Consumer and Content Producer actors in the ADX profile.

The structure of an ADX data message is quite similar to what you might already be familiar with from DXF 2 data described earlier. There are a few important differences. We will describe these differences with reference to a small example:

```
<adx xmlns="urn:ihe:qrph:adx:2015" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ihe:qrph:adx:2015 ../schema/adx_loose.xsd"
  exported="2015-02-08T19:30:00Z">
  <group orgUnit="OU_559" period="2015-06-01/P1M"
    completeDate="2015-07-01" dataSet="(TB/HIV)VCCT">
    <dataValue dataElement="VCCT_0" GENDER="FMLE" HIV_AGE="AGE0-14" value="32"/>
    <dataValue dataElement="VCCT_1" GENDER="FMLE" HIV_AGE="AGE0-14" value="20"/>
    <dataValue dataElement="VCCT_2" GENDER="FMLE" HIV_AGE="AGE0-14" value="10"/>
    <dataValue dataElement="PLHIV_TB_0" GENDER="FMLE" HIV_AGE="AGE0-14" value="10"/>
    <dataValue dataElement="PLHIV_TB_1" GENDER="FMLE" HIV_AGE="AGE0-14" value="10"/>

    <dataValue dataElement="VCCT_0" GENDER="MLE" HIV_AGE="AGE0-14" value="32"/>
    <dataValue dataElement="VCCT_1" GENDER="MLE" HIV_AGE="AGE0-14" value="20"/>
    <dataValue dataElement="VCCT_2" GENDER="MLE" HIV_AGE="AGE0-14" value="10"/>
    <dataValue dataElement="PLHIV_TB_0" GENDER="MLE" HIV_AGE="AGE0-14" value="10"/>
    <dataValue dataElement="PLHIV_TB_1" GENDER="MLE" HIV_AGE="AGE0-14" value="10"/>
```

```
    <dataValue dataElement="VCCT_0" GENDER="FMLE" HIV_AGE="AGE15-24" value="32"/>
    <dataValue dataElement="VCCT_1" GENDER="FMLE" HIV_AGE="AGE15-24" value="20"/>
    <dataValue dataElement="VCCT_2" GENDER="FMLE" HIV_AGE="AGE15-24" value="10"/>
    <dataValue dataElement="PLHIV_TB_0" GENDER="FMLE" HIV_AGE="AGE15-24" value="10"/>
    <dataValue dataElement="PLHIV_TB_1" GENDER="FMLE" HIV_AGE="AGE15-24" value="10"/>

    <dataValue dataElement="VCCT_0" GENDER="MLE" HIV_AGE="AGE15-24" value="32"/>
    <dataValue dataElement="VCCT_1" GENDER="MLE" HIV_AGE="AGE15-24" value="20"/>
    <dataValue dataElement="VCCT_2" GENDER="MLE" HIV_AGE="AGE15-24" value="10"/>
    <dataValue dataElement="PLHIV_TB_0" GENDER="MLE" HIV_AGE="AGE15-24" value="10"/>
    <dataValue dataElement="PLHIV_TB_1" GENDER="MLE" HIV_AGE="AGE15-24" value="10"/>
  </group>
</adx>
```

The adx root element

The adx root element has only one mandatory attribute, which is the *exported* timestamp. In common with other adx elements, the schema is extensible in that it does not restrict additional application specific attributes.

The group element

Unlike dxf2, adx requires that the datavalues are grouped according to orgUnit, period and dataSet. The example above shows a data report for the "(TB/HIV) VCCT" dataset from the online demo database. This example is using codes as identifiers instead of dhis2 uids. Codes are the preferred form of identifier when using adx.

The orgUnit, period and dataSet attributes are mandatory in adx. The group element may contain additional attributes. In our DHIS2 implementation any additional attributes are simply passed through to the underlying importer. This means that all attributes which currently have meaning in dxf2 (such as completeDate in the example above) can continue to be used in adx and they will be processed in the same way.

A significant difference between adx and dxf2 is in the way that periods are encoded. Adx makes strict use of ISO8601 and encodes the reporting period as (date|datetime)/(duration). So the period in the example above is a period of 1 month (P1M) starting on 2015-06-01. So it is the data for June 2015. The notation is a bit more verbose, but it is very flexible and allows us to support all existing period types in DHIS2

ADX period definitions

DHIS2 supports a limited number of periods or durations during import. Periods should begin with the date in which the duration begins, followed by a "/" and then the duration notation as noted in the table. The following table details all of the ADX supported period types, along with examples.

ADX Periods

Period type	Duration notation	Example	Duration
Daily	P1D	2017-10-01/P1M	Oct 01 2017
Weekly	P7D	2017-10-01/P7D	Oct 01 2017-Oct 07-2017
Monthly	P1M	2017-10-01/P1M	Oct 01 2017-Oct 31 2017
Bi-monthly	P2M	2017-11-01/P2M	Nov 01 2017-Dec 31 2017

Period type	Duration notation	Example	Duration
Quarterly	P3M	2017-09-01/P3M	Sep 01 2017-Dec 31 2017
Six-monthly	P6M	2017-01-01/P6M	Jan 01 2017-Jun 30 2017
Yearly	P1Ý	2017-01-01/P1Y	Jan 01 2017-Dec 31 2017
Financial October	P1Y	2017-10-01/P1Y	Oct 01 2017-Sep 30 2018
Financial April	P1Y	2017-04-01/P1Y	April 1 2017-Mar 31 2018
Financial July	P1Y	2017-07-01/P1Y	July 1 2017-June 30 2018

Data values

The dataValue element in adx is very similar to its equivalent in DXF. The mandatory attributes are *dataElement* and *value*. The *orgUnit* and *period* attributes don't appear in the dataValue as they are required at the *group* level.

The most significant difference is the way that disaggregation is represented. DXF uses the categoryOptionCombo to indicate the disaggregation of data. In adx the disaggregations (e.g. AGE*GROUP and SEX) are expressed explicitly as attributes. One important constraint on using adx is that the categories used for dataElements in the dataSet MUST have a code assigned to them, and further, that code must be of a form which is suitable for use as an XML attribute. The exact constraint on an XML attribute name is described in the W3C XML standard - in practice, this means no spaces, no non-alphanumeric characters other than '*' and it may not start with a letter. The example above shows examples of 'good' category codes ('GENDER' and 'HIV_AGE').

This restriction on the form of codes applies only to categories. Currently, the convention is not enforced by DHIS2 when you are assigning codes, but you will get an informative error message if you try to import adx data and the category codes are either not assigned or not suitable.

The main benefits of using explicit dimensions of disaggregated data are that

- The system producing the data does not have to be synchronised with the categoryOptionCombo within DHIS2.
- The producer and consumer can match their codes to a 3rd party authoritative source, such as a vterminology service. Note that in the example above the Gender and AgeGroup codes are using code lists from the WHO Global Health Observatory.

Note that this feature may be extremely useful, for example when producing disaggregated data from an EMR system, but there may be cases where a *categoryOptionCombo* mapping is easier or more desirable. The DHIS2 implementation of adx will check for the existence of a *categoryOptionCombo* attribute and, if it exists, it will use that it preference to exploded dimension attributes. Similarly, an *attributeOptionCombo* attribute on the *group* element will be processed in the legacy way. Otherwise, the attributeOptionCombo can be treated as exploded categories just as on the *dataValue*.

In the simple example above, each of the dataElements in the dataSet have the same dimensionality (categorycombo) so the data is neatly rectangular. This need not be the case. dataSets may contain dataElements with different categoryCombos, resulting in a *ragged-right* adx data message.

Importing data

DHIS2 exposes an endpoint for POST adx data at `/api/dataValueSets` using *application/xml+adx* as content type. So, for example, the following curl command can be used to POST the example data above to the DHIS2 demo server:

```
curl -u admin:district -X POST -H "Content-Type: application/adx+xml"
  -d @data.xml "https://play.dhis2.org/demo/api/33/dataValueSets?
dataElementIdScheme=code&orgUnitIdScheme=code"
```

Note the query parameters are the same as are used with DXF data. The adx endpoint should interpret all the existing DXF parameters with the same semantics as DXF.

Exporting data

DHIS2 exposes an endpoint to GET adx data sets at `/api/dataValueSets` using *application/xml+adx* as the accepted content type. So, for example, the following curl command can be used to retrieve the adx data:

```
curl -u admin:district -H "Accept: application/adx+xml"
 "https://play.dhis2.org/demo/api/33/dataValueSets?dataValueSets?
orgUnit=M_CLINIC&dataSet=MALARIA&period=201501"
```

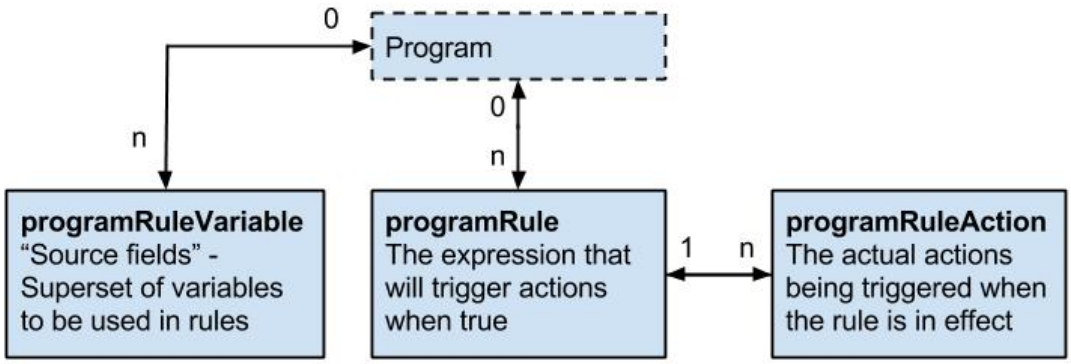
Note the query parameters are the same as are used with DXF data. An important difference is that the identifiers for dataSet and orgUnit are assumed to be codes rather than uids.

Program rules

This section is about sending and reading program rules, and explains the program rules data model. The program rules give functionality to configure dynamic behaviour in the programs in DHIS2.

Program rule model

The program rules data model consists of programRuleVariables, programRules and programRuleActions. The programRule contains an expression - when this expression is true, the child programRuleActions is triggered. The programRuleVariables is used to address data elements, tracked entity data values and other data values needed to run the expressions. All programRules in a program share the same library of programRuleVariables, and one programRuleVariable can be used in several programRules' expressions.



Program rule model details

The following table gives a detailed overview over the programRule model.

programRule

name	description	Compulsory
program	The program of which the programRule is executed in.	Compulsory
name	The name with which the program rule will be displayed to dhis2 configurators. Not visible to the end user of the program.	Compulsory
description	The description of the program rule, can be used by configurators to describe the rule. Not visible to the end user of the program.	Compulsory
programStage	If a programStage is set for a program rule, the rule will only be evaluated inside the specified program stage.	optional
condition	The expression that needs to be evaluated to true in order for the program rule to trigger its child actions. The expression is written using operators, function calls, hard coded values, constants and program rule variables. `d2:hasValue('hemoglobin') && #{hemoglobin} <= 7`	Compulsory
priority	The priority to run the rule in cases where the order of the rules matters. In most cases the rules does not depend on being run before or after other rules, and in these cases the priority can be omitted. If no priority is set, the rule will be run after any rules that has a priority defined. If a priority(integer) is set, the rule with the lowest priority will be run before rules with higher priority.	optional

Program rule action model details

The following table gives a detailed overview over the programRuleAction model.

programRuleAction

name	description	Compulsory
programRule	The programRule that is the parent of this action.	Compulsory

programRule-ActionType	The type of action that is to be performed. • `DISPLAYTEXT` - Displays a text in a given widget. • `DISPLAYKEYVALUEPAIR` - Displays a key and value pair(like a program indicator) in a given widget. • `HIDEFIELD` - Hide a specified dataElement or trackedEntityAttribute. ◦ *content* - if defined, the text in *content* will be displayed to the end user in the instance where a value is previously entered into a field that is now about to be hidden (and therefore blanked). If *content* is not defined, a standard message will be shown to the user in this instance. ◦ *dataElement* - if defined, the HIDEFIELD action will hide this dataElement when the rule is effective. ◦ *trackedEntityDataValue* - if defined, the HIDEFIELD action will hide this trackedEntityDataValue when the rule is effective. • `HIDESECTION` - Hide a specified section. ◦ *programStageSection* - must be defined. This is the programStageSection that will be hidden in case the parent rule is effective. • `ASSIGN` - Assign a dataElement a value(help the user calculate something or fill in an obvious value somewhere) ◦ *content* - if defined, the value in *data* is assigned to this variable. If content id defined, and thus a variable is assigned for use in other rules, it is important to also assign a *programRule.priority* to make sure the rule with an ASSIGN action runs before the rule that will in turn evaluate the assigned variable. ◦ *data* - must be defined, data forms an expression that is evaluated and assigned to either a variable(#{myVariable}), a dataElement, or both. ◦ *dataElement* - if defined, the value in *data* is assigned to this data element. Either the content or dataElement must be defined for the ASSIGN action to be effective. • `SHOWWARNING` - Show a warning to the user, not blocking the user from completing the event or registration. ◦ *content* - if defined, content is a static part that is displayed at the end of the error message. ◦ *data* - if defined, data forms an expression that is evaluated and added to the end of the warning message. ◦ *dataElement* - if defined, the warning message is displayed next to this data element.	Compulsory

name	description	Compulsory
location	Used for actionType DISPLAYKEYVALUEPAIR and DISPLAYTEXT to designate which widget to display the text or keyvaluepair in. Compulsory for DISPLAYKEYVALUEPAIR and DISPLAYTEXT.	See description
content	Used for user messages in the different actions. See the actionType overview for a detailed explanation for how it is used in each of the action types. Compulsory for SHOWWARNING, SHOWERROR, WARNINGONCOMPLETION, ERRORONCOMPLETION, DISPLAYTEXT and DISPLAYKEYVALUEPAIR. Optional for HIDEFIELD and ASSIGN.	See description
data	Used for expressions in the different actions. See the actionType overview for a detailed explanation for how it is used in each of the action types. Compulsory for ASSIGN. Optional for SHOWWARNING, SHOWERROR, WARNINGONCOMPLETION, ERRORONCOMPLETION, DISPLAYTEXT, CREATEEVENT and DISPLAYKEYVALUEPAIR	See description
dataElement	Used for linking rule actions to dataElements. See the actionType overview for a detailed explanation for how it is used in each of the action types. Optional for SHOWWARNING, SHOWERROR, WARNINGONCOMPLETION, ERRORONCOMPLETION, ASSIGN and HIDEFIELD	See description
trackedEntity-Attribute	Used for linking rule actions to trackedEntityAttributes. See the actionType overview for a detailed explanation for how it is used in each of the action types. Optional for SHOWWARNING, SHOWERROR and HIDEFIELD.	See description
option	Used for linking rule actions to options. See the actionType overview for a detailed explanation for how it is used in each of the action types. Optional for HIDEOPTION	See description
optionGroup	Used for linking rule actions to optionGroups. See the actionType overview for a detailed explanation for how it is used in each of the action types. Compulsory for SHOWOPTIONGROUP, HIDEOPTIONGROUP.	See description
programStage	Only used for CREATEEVENT rule actions. Compulsory for CREATEEEVENT.	See description
programStage- Section	Only used for HIDESECTION rule actions. Compulsory for HIDESECTION	See description

Program rule variable model details

The following table gives a detailed overview over the programRuleVariable model.

programRuleVariable

name	description	Compulsory
name	the name for the programRuleVariable - this name is used in expressions. `#{myVariable} > 5`	Compulsory

name	description	Compulsory
sourceType	Defines how this variable is populated with data from the enrollment and events. • DATAELEMENT_NEWEST_EVENT_PROGRAM_STAGE - In tracker capture, gets the newest value that exists for a data element, within the events of a given program stage in the current enrollment. In event capture, gets the newest value among the 10 newest events on the organisation unit. • DATAELEMENT_NEWEST_EVENT_PROGRAM - In tracker capture, get the newest value that exists for a data element across the whole enrollment. In event capture, gets the newest value among the 10 newest events on the organisation unit. • DATAELEMENT_CURRENT_EVENT - Gets the value of the given data element in the current event only. • DATAELEMENT_PREVIOUS_EVENT - In tracker capture, gets the newest value that exists among events in the program that precedes the current event. In event capture, gets the newvest value among the 10 preceeding events registered on the organisation unit. • CALCULATED_VALUE - Used to reserve a variable name that will be assigned by a ASSIGN program rule action • TEI_ATTRIBUTE - Gets the value of a given tracked entity attribute	Compulsory
dataElement	Used for linking the programRuleVariable to a dataElement. Compulsory for all sourceTypes that starts with DATAELEMENT_.	See description
trackedEntity-Attribute	Used for linking the programRuleVariable to a trackedEntityAttribute. Compulsory for sourceType TEI_ATTRIBUTE.	See description
useCodeFor-OptionSet	If checked, the variable will be populated with the code - not the name - from any linked option set. Default is unchecked, meaning that the name of the option is populated.	
programStage	Used for specifying a specific program stage to retreive the programRuleVariable value from. Compulsory for DATAELEMENT_NEWEST_EVENT_PROGRAM_STAGE.	See description

Creating program rules

- To perform crud operations, `programRules` resource is available in API.

To retrieve list of programRules you can do a GET request like this:

```
/api/programRules
```

To retrieve single programRule you can do a GET request like this:

```
/api/programRules/<program_rule_uid>
```

To save/add single programRule you can do a POST request like this:

```
/api/programRules/<program_rule_uid>
```

To update single programRule you can do a PUT request like this:

```
/api/programRules/<program_rule_uid>
```

To delete single programRule you can do a DELETE request like this:

```
/api/programRules/<program_rule_uid>
```

To retrieve description of programRule condition you can use POST and provide condition string in the POST body.

```
/api/programRules/condition/description?<program_rule_uid>
```

Forms

To retrieve information about a form (which corresponds to a data set and its sections) you can interact with the `form` resource. The form response is accessible as XML and JSON and will provide information about each section (group) in the form as well as each field in the sections, including labels and identifiers. By supplying period and organisation unit identifiers the form response will be populated with data values.

Form query parameters

Parameter	Option	Description
pe	ISO period	Period for which to populate form data values.
ou	UID	Organisation unit for which to populate form data values.
metaData	false | true	Whether to include metadata about each data element of form sections.

To retrieve the form for a data set you can do a GET request like this:

```
/api/dataSets/<dataset-id>/form.json
```

To retrieve the form for the data set with identifier "BfMAe6Itzgt" in XML:

```
/api/dataSets/BfMAe6Itzgt/form
```

To retrieve the form including metadata in JSON:

```
/api/dataSets/BfMAe6Itzgt/form.json?metaData=true
```

To retrieve the form filled with data values for a specific period and organisation unit in XML:

```
/api/dataSets/BfMAe6Itzgt/form.xml?ou=DiszpKrYNg8&pe=201401
```

When it comes to custom data entry forms, this resource also allows for creating such forms directly for a data set. This can be done through a POST or PUT request with content type text/html where the payload is the custom form markup such as:

```
curl -d @form.html "localhost/api/dataSets/BfMAe6Itzgt/form"
  -H "Content-Type:text/html" -u admin:district -X PUT
```

Documents

References to files can be stored with the document resource.

Document fields

Field name	Description
name	unique name of document
external	flag identifying the location of the document. TRUE for external files, FALSE for internal ones
url	the location of the file. URL for external files. File resource id for internal ones (see File resources)

A GET request to the documents endpoint will return all documents:

```
/api/documents
```

A POST request to the documents endpoint will create a new document:

```
curl -X POST -d @document.json -H "Content-type: application/json"
  "http://dhis.domain/api/documents"
```

```
{
    "name": "dhis home",
    "external": true,
    "url": "https://www.dhis2.org"
}
```

A GET request with the id of a document appended will return information about the document. A PUT request to the same endpoint will update the fields of the document:

```
/api/documents/<documentId>
```

Appending */data* to the GET request will return the actual file content of the document:

```
/api/documents/<documentId>/data
```

Validation

To generate a data validation summary you can interact with the validation resource. The dataSet resource is optimized for data entry clients for validating a data set / form, and can be accessed like this:

```
/api/33/validation/dataSet/QX4ZTUbOt3a.json?pe=201501&ou=DiszpKrYNg8
```

In addition to validate rules based on data set, there are two additional methods for performing validation: Custom validation and Scheduled validation.

Custom validation can be initiated through the "Data Quality" app, where you can configure the periods, validation rule groups and organisation units to be included in the analysis and if you want to send out notifications for and/or persist the results found. The result of this analysis will be a list of violations found using your criteria.

The first path variable is an identifier referring to the data set to validate. XML and JSON resource representations are supported. The response contains violations of validation rules. This will be extended with more validation types in the coming versions.

To retrieve validation rules which are relevant for a specific data set, meaning validation rules with formulas where all data elements are part of the specific data set, you can make a GET request to to `validationRules` resource like this:

```
/api/validationRules?dataSet=<dataset-id>
```

The validation rules have a left side and a right side, which is compared for validity according to an operator. The valid operator values are found in the table below.

Operators

Value	Description
equal_to	Equal to
not_equal_to	Not equal to
greater_than	Greater than
greater_than_or_equal_to	Greater than or equal to
less_than	Less than
less_than_or_equal_to	Less than or equal to
compulsory_pair	If either side is present, the other must also be
exclusive_pair	If either side is present, the other must not be

The left side and right side expressions are mathematical expressions which can contain references to data elements and category option combinations on the following format:

```
${<dataelement-id>.<catoptcombo-id>}
```

The left side and right side expressions have a *missing value strategy*. This refers to how the system should treat data values which are missing for data elements / category option combination references in the formula in terms of whether the validation rule should be checked for validity or skipped. The valid missing value strategies are found in the table below.

Missing value strategies

Value	Description
SKIP_IF_ANY_VALUE_MISSING	Skip validation rule if any data value is missing
SKIP_IF_ALL_VALUES_MISSING	Skip validation rule if all data values are missing
NEVER_SKIP	Never skip validation rule irrespective of missing data values

Validation Results

Validation results are persisted results of violations found during a validation analysis. If you choose "persist results" when starting or scheduling a validation analysis, any violations found will be stored in the database. When a result is stored in the database it will be used for 3 things:

1. Generating analytics based on the stored results.
2. Persisted results that have not generated a notification, will do so, once.
3. Keeping track of whether or not the result has generated a notification.
4. Skipping rules that have been already checked when running validation analysis.

This means if you don't persist your results, you will be unable to generate analytics for validation results, if checked, results will generate notifications every time it's found and running validation analysis might be slower.

The validation results persisted can be viewed at the following endpoint:

```
/api/33/validationResults
```

You can also inspect an individual result using the validation result id in this endpoint:

```
/api/33/validationResults/<id>
```

Validation results are sent out to the appropriate users once every day, but can also be manually triggered to run on demand using the following api endpoint:

```
/api/33/validation/sendNotifications
```

Only unsent results are sent using this endpoint.

Data analysis

Several resources for performing data analysis and finding data quality and validation issues are provided.

Validation rule analysis

To run validation rules and retrieve violations:

```
/api/dataAnalysis/validationRules
```

The following query parameters are supported:

Validation rule analysis query parameters

Query parameter	Description	Option
vrg	Validation rule group	ID
ou	Organisation unit	ID
startDate	Start date for the timespan	Date
endDate	End date for the timespan	Date
persist	Whether to persist violations in the system	false | true
notification	Whether to send notifications about violations	false | true

Sample output:


```
[
    {
        "validationRuleId": "kgh54Xb9LSE",
        "validationRuleDescription": "Malaria outbreak",
        "organisationUnitId": "DiszpKrYNg8",
        "organisationUnitDisplayName": "Ngelehun CHC",
        "organisationUnitPath": "/ImspTQPwCqd/O6uvpzGd5pu/YuQRtpLP10I/DiszpKrYNg8",
        "organisationUnitAncestorNames": "Sierra Leone / Bo / Badjia / ",
        "periodId": "201901",
        "periodDisplayName": "January 2019",
        "attributeOptionComboId": "HllvX50cXC0",
        "attributeOptionComboDisplayName": "default",
        "importance": "MEDIUM",
        "leftSideValue": 10.0,
        "operator": ">",
        "rightSideValue": 14.0
    },
    {
        "validationRuleId": "ZoG4yXZi3c3",
        "validationRuleDescription": "ANC 2 cannot be higher than ANC 1",
        "organisationUnitId": "DiszpKrYNg8",
        "organisationUnitDisplayName": "Ngelehun CHC",
        "organisationUnitPath": "/ImspTQPwCqd/O6uvpzGd5pu/YuQRtpLP10I/DiszpKrYNg8",
        "organisationUnitAncestorNames": "Sierra Leone / Bo / Badjia / ",
        "periodId": "201901",
        "periodDisplayName": "January 2019",
        "attributeOptionComboId": "HllvX50cXC0",
        "attributeOptionComboDisplayName": "default",
        "importance": "MEDIUM",
        "leftSideValue": 22.0,
        "operator": "<=",
        "rightSideValue": 19.0
    }
]
```


Standard deviation based outlier analysis

To identify data outliers based on standard deviations of the average value:

```
/api/dataAnalysis/stdDevOutlier
```

The following query parameters are supported:

Standard deviation outlier analysis query parameters

Query parameter	Description	Option
ou	Organisation unit	ID
startDate	Start date for the timespan	Date
endDate	End date for the timespan	Date
ds	Data sets, parameter can be repeated	ID
standardDeviation	Number of standard deviations from the average	Numeric value

Min/max value based outlier analysis

To identify data outliers based on min/max values:

```
/api/dataAnalysis/minMaxOutlier
```

The supported query parameters are equal to the *std dev based outlier analysis* resource described above.

Follow-up data analysis

To identify data marked for follow-up:

```
/api/dataAnalysis/followup
```

The supported query parameters are equal to the *std dev based outlier analysis* resource described above.

Data integrity

The data integrity capabilities of the data administration module are available through the web API. This section describes how to run the data integrity process as well as retrieving the result. The details of the analysis performed are described in the user manual.

Running data integrity

The operation of measuring data integrity is a fairly resource (and time) demanding task. It is therefore run as an asynchronous process and only when explicitly requested. Starting the task is done by forming an empty POST request to the *dataIntegrity* endpoint like so (demonstrated in curl syntax):

```
curl -X POST "https://localhost/api/33/dataIntegrity"
```

If successful the request will return HTTP 202 immediately. The location header of the response points to the resource used to check the status of the request. The payload also contains a json

object of the job created. Forming a GET request to the given location yields an empty JSON response if the task has not yet completed and a JSON taskSummary object when the task is done. Polling (conservatively) to this resource can hence be used to wait for the task to finish.

Fetching the result

Once data integrity is finished running the result can be fetched from the `system/taskSummaries` resource like so:

```
curl "https://dhis.domain/api/33/system/taskSummaries/DATAINTEGRITY"
```

The returned object contains a summary for each point of analysis, listing the names of the relevant integrity violations. As stated in the leading paragraph for this section the details of the analysis (and the resulting data) can be found in the user manual chapter on Data Administration.

Indicators

This section describes indicators and indicator expressions.

Aggregate indicators

To retrieve indicators you can make a GET request to the indicators resource like this:

```
/api/indicators
```

Indicators represent expressions which can be calculated and presented as a result. The indicator expressions are split into a numerator and denominator. The numerators and denominators are mathematical expressions which can contain references to data elements, other indicators, constants and organisation unit groups. The variables will be substituted with data values when used e.g. in reports. Variables which are allowed in expressions are described in the following table.

Indicator variables

Variable	Object	Description
#{<dataelement-id>.<categoryoptcombo-id>.<attributeoptcombo-id>}	Data element operand	Refers to a combination of an aggregate data element and a category option combination. Both category and attribute option combo ids are optional, and a wildcard "*" symbol can be used to indicate any value.
#{<dataelement-id>.<categoryoptiongroup-id>.<attributeoptcombo-id>}	Category Option Group	Refers to an aggregate data element and a category option group, containing multiple category option combinations.
#{<dataelement-id>}	Aggregate data element	Refers to the total value of an aggregate data element across all category option combinations.
D{<program-id>.<dataelement-id>}	Program data element	Refers to the value of a tracker data element within a program.
A{<program-id>.<attribute-id>}	Program tracked entity attribute	Refers to the value of a tracked entity attribute within a program.

Variable	Object	Description
I{<program-indicator-id>}	Program indicator	Refers to the value of a program indicator.
R{<dataset-id>.<metric>}	Reporting rate	Refers to a reporting rate metric. The metric can be REPORTING_RATE, REPORTING_RATE_ON_TIME, ACTUAL_REPORTS, ACTUAL_REPORTS_ON_TIME, EXPECTED_REPORTS.
C{<constant-id>}	Constant	Refers to a constant value.
N{<indicator-id>}	Indicator	Refers to an existing Indicator.
OUG{<orgunitgroup-id>}	Organisation unit group	Refers to the count of organisation units within an organisation unit group.

The syntax looks like this:

```
#{<dataelement-id>.<catoptcombo-id>} + C{<constant-id>} + OUG{<orgunitgroup-id>}
```

A corresponding example looks like this:

```
#{P3jJH5Tu5VC.S34ULMcHMca} + C{Gfd3ppDfq8E} + OUG{CXw2yu5fodb}
```

Note that for data element variables the category option combo identifier can be omitted. The variable will then represent the total for the data element, e.g. across all category option combos. Example:

```
#{P3jJH5Tu5VC} + 2
```

Data element operands can include any of category option combination and attribute option combination, and use wildcards to indicate any value:

```
#{P3jJH5Tu5VC.S34ULMcHMca} + #{P3jJH5Tu5VC.*.j8vBiBqGf6O} + #{P3jJH5Tu5VC.S34ULMcHMca.*}
```

An example which uses a program data element and a program attribute:

```
( D{eBAyeGv0exc.vV9UWAZohSf} * A{IpHINAT79UW.cejWyOfXge6} ) / D{eBAyeGv0exc.GieVkTxp4HH}
```

An example which combines program indicators and aggregate indicators:

```
I{EMOt6Fwhs1n} * 1000 / #{WUg3MYWQ7pt}
```

An example which uses a reporting rate looks like this:

```
R{BfMAe6Itzgt.REPORTING_RATE} * #{P3jJH5Tu5VC.S34ULMcHMca}
```

Another example which uses actual data set reports:

```
R{BfMAe6Itzgt.ACTUAL_REPORTS} / R{BfMAe6Itzgt.EXPECTED_REPORTS}
```

An example which uses an existing indicator would look like this:

```
N{Rigf2d2Zbjp} * #{P3jJH5Tu5VC.S34ULMcHMca}
```

Expressions can be any kind of valid mathematical expression, as an example:

```
( 2 * #{P3jJH5Tu5VC.S34ULMcHMca} ) / ( #{FQ2o8UBlcrS.S34ULMcHMca} - 200 ) * 25
```

Program indicators

To retrieve program indicators you can make a GET request to the program indicators resource like this:

```
/api/programIndicators
```

Program indicators can contain information collected in a program. Indicators have an expression which can contain references to data elements, attributes, constants and program variables. Variables which are allowed in expressions are described in the following table.

Program indicator variables

Variable	Description
#{<programstage-id>.<dataelement-id>}	Refers to a combination of program stage and data element id.
A{<attribute-id>}	Refers to a tracked entity attribute.
V{<variable-id>}	Refers to a program variable.
C{<constant-id>}	Refers to a constant.

The syntax looks like this:

```
#{<programstage-id>.<dataelement-id>} + #{<attribute-id>} + V{<varible-id>} + C{<constant-id>}
```

A corresponding example looks like this:

```
#{A03MvHHogjR.a3kGcGDCuk6} + A{OvY4VVhSDeJ} + V{incident_date} + C{bCqvfPR02Im}
```

Expressions

Expressions are mathematical formulas which can contain references to data elements, constants and organisation unit groups. To validate and get the textual description of an expression, you can make a GET request to the expressions resource:

```
/api/expressions/description?expression=<expression-string>
```

The response follows the standard JSON web message format. The *status* property indicates the outcome of the validation and will be "OK" if successful and "ERROR" if failed. The *message* property will be "Valid" if successful and provide a textual description of the reason why the validation failed if not. The *description* provides a textual description of the expression.

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Valid",
    "description": "Acute Flaccid Paralysis"
}
```

Complete data set registrations

This section is about complete data set registrations for data sets. A registration marks as a data set as completely captured.

Completing data sets

This section explains how to register data sets as complete. This is achieved by interacting with the *completeDataSetRegistrations* resource:

```
/api/33/completeDataSetRegistrations
```

The endpoint supports the *POST* method for registering data set completions. The endpoint is functionally very similar to the *dataValueSets* endpoint, with support for bulk import of complete registrations.

Importing both *XML* and *JSON* formatted payloads are supported. The basic format of this payload, given as *XML* in this example, is like so:

```
<completeDataSetRegistrations xmlns="http://dhis2.org/schema/dxf/2.0">
  <completeDataSetRegistration period="200810" dataSet="eZDhcZi6FLP"
    organisationUnit="qhqAxPSTUXp" attributeOptionCombo="bRowv6yZOF2" storedBy="imported"/>
  <completeDataSetRegistration period="200811" dataSet="eZDhcZi6FLP"
    organisationUnit="qhqAxPSTUXp" attributeOptionCombo="bRowv6yZOF2" storedBy="imported"/>
</completeDataSetRegistrations>
```

The *storedBy* attribute is optional (as it is a nullable property on the complete registration object). You can also optionally set the *date* property (time of registration) as an attribute. It the time is not set, the current time will be used.

The import process supports the following query parameters:

Complete data set registrations query parameters

Parameter	Values	Description
dataSetIdScheme	id | name | code | attribute:ID	Property of the data set to use to map the complete registrations.
orgUnitIdScheme	id | name | code | attribute:ID	Property of the organisation unit to use to map the complete registrations.
attributeOptionComboIdScheme	id | name | code | attribute:ID	Property of the attribute option combos to use to map the complete registrations.
idScheme	id | name | code | attribute:ID	Property of all objects including data sets, org units and attribute option combos, to use to map the complete registrations.
preheatCache	false | true	Whether to save changes on the server or just return the import summary.
dryRun	false | true	Whether registration applies to sub units
importStrategy	CREATE | UPDATE | CREATE_AND_UPDATE | DELETE	Save objects of all, new or update import status on the server.
skipExistingCheck	false | true	Skip checks for existing complete registrations. Improves performance. Only use for empty databases or when the registrations to import do not exist already.
async	false | true	Indicates whether the import should be done asynchronous or synchronous. The former is suitable for very large imports as it ensures that the request does not time out, although it has a significant performance overhead. The latter is faster but requires the connection to persist until the process is finished.

Reading complete data set registrations

This section explains how to retrieve data set completeness registrations. We will be using the *completeDataSetRegistrations* resource. The query parameters to use are these:

Data value set query parameters

Parameter	Description
dataSet	Data set identifier, multiple data sets are allowed
period	Period identifier in ISO format. Multiple periods are allowed.
startDate	Start date for the time span of the values to export
endDate	End date for the time span of the values to export
created	Include only registrations which were created since the given timestamp
createdDuration	Include only registrations which were created within the given duration. The format is <value><time-unit>, where the supported time units are "d", "h", "m", "s" *(days, hours, minutes, seconds).* The time unit is relative to the current time.
orgUnit	Organisation unit identifier, can be specified multiple times. Not applicable if orgUnitGroup is given.
orgUnitGroup	Organisation unit group identifier, can be specified multiple times. Not applicable if orgUnit is given.

Parameter	Description
children	Whether to include the children in the hierarchy of the organisation units
limit	The maximum number of registrations to include in the response.
idScheme	Identifier property used for meta data objects in the response.
dataSetIdScheme	Identifier property used for data sets in the response. Overrides idScheme.
orgUnitIdScheme	Identifier property used for organisation units in the response. Overrides idScheme.
attributeOptionComboIdScheme	Identifier property used for attribute option combos in the response. Overrides idScheme.

The dataSet and orgUnit parameters can be repeated in order to include multiple data sets and organisation units.

The period, start/end date, created and createdDuration parameters provide multiple ways to set the time dimension for the request, thus only one can be used. For example, it doesn't make sense to both set the start/end date and to set the periods.

An example request looks like this:

```
curl "https://play.dhis2.org/demo/api/33/completeDataSetRegistrations?dataSet=pBOMPrpg1QX
  &dataSet=pBOMPrpg1QX&startDate=2014-01-01&endDate=2014-01-31&orgUnit=YuQRtpLP10I
  &orgUnit=vWbkYPRmKyS&children=true"
  -H "Accept:application/xml" -u admin:district
```

You can get the response in *xml* and *json* format. You can indicate which response format you prefer through the *Accept* HTTP header like in the example above. For xml you use *application/xml*; for json you use *application/json*.

Un-completing data sets

This section explains how you can un-register the completeness of a data set. To un-complete a data set you will interact with the completeDataSetRegistrations resource:

```
/api/33/completeDataSetRegistrations
```

This resource supports *DELETE* for un-registration. The following query parameters are supported:

Complete data set registrations query parameters

Query parameter	Required	Description
ds	Yes	Data set identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
cc	No (must combine with cp)	Attribute combo identifier (for locking check)

Query parameter	Required	Description
cp	No (must combine with cp)	Attribute option identifiers, separated with ; for multiple values (for locking check)
multiOu	No (default false)	Whether registration applies to sub units

Data approval

This section explains how to approve, unapprove and check approval status using the *dataApprovals* resource. Approval is done per data approval workflow, period, organisation unit and attribute option combo.

```
/api/33/dataApprovals
```

A data approval workflow is associated with several entities:

- A period type which defines the frequency of approval
- An optional category combination
- One or many data approval levels which are part of the workflow
- One or many data sets which are used for data collection

Get approval status

To get approval information for a data set you can issue a GET request:

```
/api/dataApprovals?wf=rIUL3hYOjJc&pe=201801&ou=YuQRtpLP10I
```

Data approval query parameters

Query parameter	Required	Description
wf	Yes	Data approval workflow identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
aoc	No	Attribute option combination identifier

> **Note**
>
> For backward compatibility, the parameter `ds` for data set may be given instead of `wf` for workflow in this and other data approval requests as described below. If the data set is given, the workflow associated with that data set will be used.

This will produce a response similar to this:

```
{
    "mayApprove": false,
    "mayUnapprove": false,
```

```
    "mayAccept": false,
    "mayUnaccept": false,
    "state": "UNAPPROVED_ELSEWHERE"
}
```

The returned parameters are:

Data approval returned parameters

Return Parameter	Description
mayApprove	Whether the current user may approve this data selection.
mayUnapprove	Whether the current user may unapprove this data selection.
mayAccept	Whether the current user may accept this data selection.
mayUnaccept	Whether the current user may unaccept this data selection.
state	One of the data approval states from the table below.

Data approval states

State	Description
UNAPPROVABLE	Data approval does not apply to this selection. (Data is neither approved nor unapproved.)
UNAPPROVED_WAITING	Data could be approved for this selection, but is waiting for some lower-level approval before it is ready to be approved.
UNAPPROVED_ELSEWHERE	Data is unapproved, and is waiting for approval somewhere else (not approvable here.)
UNAPPROVED_READY	Data is unapproved, and is ready to be approved for this selection.
APPROVED_HERE	Data is approved, and was approved here (so could be unapproved here.)
APPROVED_ELSEWHERE	Data is approved, but was not approved here (so cannot be unapproved here.) This covers the following cases: • Data is approved at a higher level. • Data is approved for wider scope of category options. • Data is approved for all sub-periods in selected period. In the first two cases, there is a single data approval object that covers the selection. In the third case there is not.
ACCEPTED_HERE	Data is approved and accepted here (so could be unapproved here.)
ACCEPTED_ELSEWHERE	Data is approved and accepted, but elsewhere.

Note that when querying for the status of data approval, you may specify any combination of the query parameters. The combination you specify does not need to describe the place where data is to be approved at one of the approval levels. For example:

- The organisation unit might not be at an approval level. The approval status is determined by whether data is approved at an approval level for an ancestor of the organisation unit.
- You may specify individual attribute category options. The approval status is determined by whether data is approved for an attribute category option combination that includes one or more of these options.

- You may specify a time period that is longer than the period for the data set at which the data is entered and approved. The approval status is determined by whether the data is approved for all the data set periods within the period you specify.

For data sets which are associated with a category combo you might want to fetch data approval records for individual attribute option combos from the following resource with a GET request:

```
/api/dataApprovals/categoryOptionCombos?wf=rIUL3hYOjJc&pe=201801&ou=YuQRtpLP10I
```

Bulk get approval status

To get a list of multiple approval statuses, you can issue a GET request similar to this:

```
/api/dataApprovals/approvals?wf=rIUL3hYOjJc&pe=201801,201802&ou=YuQRtpLP10I
```

The parameters `wf`, `pe`, `ou`, and `aoc` are the same as for getting a single approval status, except that you can provide a comma-separated list of one or more values for each parameter.

This will give you a response containing a list of approval parameters and statuses, something like this:

```
[
    {
        "aoc": "HllvX50cXC0",
        "pe": "201801",
        "level": "KaTJLhGmU95",
        "ou": "YuQRtpLP10I",
        "permissions": {
            "mayApprove": false,
            "mayUnapprove": true,
            "mayAccept": true,
            "mayUnaccept": false,
            "mayReadData": true
        },
        "state": "APPROVED_HERE",
        "wf": "rIUL3hYOjJc"
    },
    {
        "aoc": "HllvX50cXC0",
        "pe": "201802",
        "ou": "YuQRtpLP10I",
        "permissions": {
            "mayApprove": true,
            "mayUnapprove": false,
            "mayAccept": false,
            "mayUnaccept": false,
            "mayReadData": true
        },
        "state": "UNAPPROVED_READY",
        "wf": "rIUL3hYOjJc"
    }
]
```

The returned fields are described in the table below.

Field	Description
aoc	Attribute option combination identifier
pe	Period identifier
ou	Organisation Unit identifier
permissions	The permissions: 'mayApprove', 'mayUnapprove', 'mayAccept', 'mayUnaccept', and 'mayReadData' (same definitions as for get single approval status).
state	One of the data approval states (same as for get single approval status.)
wf	Data approval workflow identifier

Approve data

To approve data you can issue a *POST* request to the *dataApprovals* resource. To un-approve data, you can issue a *DELETE* request to the dataApprovals resource.

```
POST DELETE /api/33/dataApprovals
```

To accept data that is already approved you can issue a *POST* request to the *dataAcceptances* resource. To un-accept data, you can issue a *DELETE* request to the *dataAcceptances* resource.

```
POST DELETE /api/33/dataAcceptances
```

These requests contain the following parameters:

Data approval action parameters

Action parameter	Required	Description
wf	Yes	Data approval workflow identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
aoc	No	Attribute option combination identifier

Note that, unlike querying the data approval status, you must specify parameters that correspond to a selection of data that could be approved. In particular, both of the following must be true:

- The organisation unit's level must be specified by an approval level in the workflow.
- The time period specified must match the period type of the workflow.

Bulk approve data

You can approve a bulk of data records by posting to the `/api/dataApprovals/approvals` resource.

```
POST /api/33/dataApprovals/approvals
```

You can unapprove a bulk of data records by posting to the `/api/dataApprovals/unapprovals` resource.

```
POST /api/33/dataApprovals/unapprovals
```

You can accept a bulk of records by posting to the `/api/dataAcceptances/acceptances` resource.

```
POST /api/33/dataAcceptances/acceptances
```

You can unaccept a bulk of records by posting to the `/api/dataAcceptances/unacceptances` resource.

```
POST /api/33/dataAcceptances/unacceptances
```

The approval payload is supported as JSON and looks like this:

```
{
    "wf": ["pBOMPrpg1QX", "lyLU2wR22tC"],
    "pe": ["201601", "201602"],
    "approvals": [
        {
            "ou": "cDw53Ej8rju",
            "aoc": "ranftQIH5M9"
        },
        {
            "ou": "cDw53Ej8rju",
            "aoc": "fC3z1lcAW5x"
        }
    ]
}
```

Get data approval levels

To retrieve data approval workflows and their data approval levels you can make a GET request similar to this:

```
/api/dataApprovalWorkflows?
  fields=id,name,periodType,dataApprovalLevels[id,name,level,orgUnitLevel]
```

Auditing

DHIS2 does automatic auditing on all updates and deletions of aggregate data values, tracked entity data values, tracked entity attribute values, and data approvals. This section explains how to fetch this data.

Aggregate data value audits

The endpoint for aggregate data value audits is located at `/api/audits/dataValue`, and the available parameters are displayed in the table below.

Aggregate data value query parameters

Parameter	Option	Description
ds	Data Set	One or more data set identifiers to get data elements from.
de	Data Element	One or more data element identifiers.
pe	ISO Period	One or more period ISO identifiers.
ou	Organisation Unit	One or more org unit identifiers.
auditType	UPDATE | DELETE	Filter by audit type.
skipPaging	false | true	Turn paging on / off
page	1 (default)	If paging is enabled, this parameter decides which page to show

Get all audits for data set with ID *lyLU2wR22tC*:

```
/api/33/audits/dataValue?ds=lyLU2wR22tC
```

Tracked entity data value audits

The endpoint for tracked entity data value audits is located at `/api/audits/trackedEntityDataValue`, and the available parameters are displayed in the table below.

Tracked entity data value query parameters

Parameter	Option	Description
de	Data Element	One or more data element identifiers.
ps	Program Stage Entity	One or more program stage instance identifiers.
auditType	UPDATE | DELETE	Filter by audit type.
skipPaging	false | true	Turn paging on / off
page	1 (default)	If paging is enabled, this parameter decides which page to show

Get all audits which have data element ID eMyVanycQSC or qrur9Dvnyt5:

```
/api/33/audits/trackedEntityDataValue?de=eMyVanycQSC&de=qrur9Dvnyt5
```

Tracked entity attribute value audits

The endpoint for tracked entity attribute value audits is located at `/api/audits/trackedEntityAttributeValue`, and the available parameters are displayed in the table below.

Tracked entity attribute value query parameters

Parameter	Option	Description
tea	Tracked Entity Attributes	One or more tracked entity attribute identifiers.
te	Tracked Entity Instances	One or more tracked entity instance identifiers.
auditType	UPDATE | DELETE	Filter by audit type.
skipPaging	false | true	Turn paging on / off
page	1 (default)	If paging is enabled, this parameter decides which page to show

Get all audits which have attribute with ID VqEFza8wbwA:

```
/api/33/audits/trackedEntityAttributeValue?tea=VqEFza8wbwA
```

Tracked entity instance audits

Once auditing is enabled for tracked entity instances (by setting allowAuditLog of tracked entity types to true), all read and search operations are logged. The endpoint for accessing audit logs is api/audits/trackedEntityInstance. Below are available parameters to interact with this endpoint.

Tracked entity instance audit query parameters

Parameter	Option	Description
tei	Tracked Entity Instance	One or more tracked entity instance identifiers
user	User	One or more user identifiers
auditType	SEARCH | READ	Audit type to filter for
startDate	Start date	Start date for audit filtering in yyyy-mm-dd format.
endDate	End date	End date for audit filtering in yyyy-mm-dd format.
skipPaging	false | true	Turn paging on / off.
page	1 (default)	Specific page to ask for.
pageSize	50 (default)	Page size.

Get all tracked entity instance audits of type READ with startDate=2018-03-01 and endDate=2018-04-24 in a page size of 5:

```
/api/33/audits/trackedEntityInstance.json?startDate=2018-03-01
  &endDate=2018-04-24&auditType=READ&pageSize=5
```

Enrollment audits

Once auditing is enabled for enrollments (by setting allowAuditLog of tracker programs to true), all read operations are logged. The endpoint for accessing audit logs is api/audits/enrollment. Below are available parameters to interact with this endpoint.

Enrollment audit query parameters

Parameter	Option	Description
en	Enrollment	One or more tracked entity instance identifiers
user	User	One or more user identifiers
startDate	Start date	Start date for audit filtering in yyyy-mm-dd format.
endDate	End date	End date for audit filtering in yyyy-mm-dd format.
skipPaging	false | true	Turn paging on / off.
page	1 (default)	Specific page to ask for.
pageSize	50 (default)	Page size.

Get all enrollment audits with startDate=2018-03-01 and endDate=2018-04-24 in a page size of 5:

```
/api/audits/enrollment.json?startDate=2018-03-01&endDate=2018-04-24&pageSize=5
```

Get all enrollment audits for user admin:

```
/api/audits/enrollment.json?user=admin
```

Data approval audits

The endpoint for data approval audits is located at /api/audits/dataApproval, and the available parameters are displayed in the table below.

Data approval query parameters

Parameter	Option	Description
dal	Data Approval Level	One or more data approval level identifiers.
wf	Workflow	One or more data approval workflow identifiers.
ou	Organisation Unit	One or more organisation unit identifiers.
aoc	Attribute Option Combo	One or more attribute option combination identifiers.
startDate	Start Date	Starting Date for approvals in yyyy-mm-dd format.
endDate	End Date	Ending Date for approvals in yyyy-mm-dd format.
skipPaging	false | true	Turn paging on / off
page	1 (default)	If paging is enabled, this parameter decides which page to show.

Get all audits for data approval workflow RwNpkAM7Hw7:

```
/api/33/audits/dataApproval?wf=RwNpkAM7Hw7
```

Message conversations

DHIS2 features a mechanism for sending messages for purposes such as user feedback, notifications, and general information to users. Messages are grouped into conversations. To interact with message conversations you can send POST and GET request to the *messageConversations* resource.

```
/api/33/messageConversations
```

Messages are delivered to the DHIS2 message inbox but can also be sent to the user's email addresses and mobile phones as SMS. In this example, we will see how we can utilize the Web API to send, read and manage messages. We will pretend to be the *DHIS2 Administrator* user and send a message to the *Mobile* user. We will then pretend to be the mobile user and read our new message. Following this, we will manage the admin user inbox by marking and removing messages.

Writing and reading messages

The resource we need to interact with when sending and reading messages is the *messageConversations* resource. We start by visiting the Web API entry point at http://play.dhis2.org/demo/api where we find and follow the link to the *messageConversations* resource at http://play.dhis2.org/demo/api/messageConversations. The description tells us that we can use a POST request to create a new message using the following XML format for sending to multiple users:

```
<message xmlns="http://dhis2.org/schema/dxf/2.0">
  <subject>This is the subject</subject>
  <text>This is the text</text>
  <users>
    <user id="user1ID" />
    <user id="user2ID" />
    <user id="user3ID" />
  </users>
</message>
```

For sending to all users contained in one or more user groups, we can use:

```
<message xmlns="http://dhis2.org/schema/dxf/2.0">
  <subject>This is the subject</subject>
  <text>This is the text</text>
  <userGroups>
    <userGroup id="userGroup1ID" />
    <userGroup id="userGroup2ID" />
    <userGroup id="userGroup3ID" />
  </userGroups>
</message>
```

For sending to all users connected to one or more organisation units, we can use:

```
<message xmlns="http://dhis2.org/schema/dxf/2.0">
  <subject>This is the subject</subject>
  <text>This is the text</text>
  <organisationUnits>
    <organisationUnit id="ou1ID" />
    <organisationUnit id="ou2ID" />
    <organisationUnit id="ou3ID" />
  </organisationUnits>
</message>
```

Since we want to send a message to our friend the mobile user we need to look up her identifier. We do so by going to the Web API entry point and follow the link to the *users* resource at `/api/users`. We continue by following link to the mobile user at `/api/users/PhzytPW3g2J` where we learn that her identifier is *PhzytPW3g2J*. We are now ready to put our XML message together to form a message where we want to ask the mobile user whether she has reported data for January 2014:

```
<message xmlns="http://dhis2.org/schema/dxf/2.0">
  <subject>Mortality data reporting</subject>
  <text>Have you reported data for the Mortality data set for January 2014?</text>
  <users>
    <user id="PhzytPW3g2J" />
  </users>
</message>
```

To test this we save the XML content into a file called *message.xml*. We use cURL to dispatch the message the DHIS2 demo instance where we indicate that the content-type is XML and authenticate as the *admin* user:

```
curl -d @message.xml "https://play.dhis2.org/demo/api/messageConversations"
  -H "Content-Type:application/xml" -u admin:district -X POST
```

A corresponding payload in JSON and POST command looks like this:

```
{
    "subject": "Hey",
    "text": "How are you?",
    "users": [
        {
            "id": "OYLGMiazHtW"
        },
        {
            "id": "N3PZBUlN8vq"
        }
    ],
    "userGroups": [
        {
            "id": "ZoHNWQajIoe"
        }
    ],
    "organisationUnits": [
        {
            "id": "DiszpKrYNg8"
        }
    ]
}
```

```
curl -d @message.json "https://play.dhis2.org/demo/api/33/messageConversations"
  -H "Content-Type:application/json" -u admin:district -X POST
```

If all is well we receive a *201 Created* HTTP status code. Also, note that we receive a *Location* HTTP header which value informs us of the URL of the newly created message conversation resource - this can be used by a consumer to perform further action.

We will now pretend to be the mobile user and read the message which was just sent by dispatching a GET request to the *messageConversations* resource. We supply an *Accept* header with *application/xml* as the value to indicate that we are interested in the XML resource representation and we authenticate as the *mobile* user:

```
curl "https://play.dhis2.org/demo/api/33/messageConversations"
  -H "Accept:application/xml" -u mobile:district
```

In response we get the following XML:

```
<messageConversations xmlns="http://dhis2.org/schema/dxf/2.0"
  link="https://play.dhis2.org/demo/api/messageConversations">
  <messageConversation name="Mortality data reporting" id="ZjHHSjyyeJ2"
    link="https://play.dhis2.org/demo/api/messageConversations/ZjHHSjyyeJ2"/>
  <messageConversation name="DHIS2 version 2.7 is deployed" id="GDBqVfkmnp2"
    link="https://play.dhis2.org/demo/api/messageConversations/GDBqVfkmnp2"/>
</messageConversations>
```

From the response, we are able to read the identifier of the newly sent message which is *ZjHHSjyyeJ2*. Note that the link to the specific resource is embedded and can be followed in order to read the full message. We can reply directly to an existing message conversation once we know the URL by including the message text as the request payload. We are now able to construct a URL for sending our reply:

```
curl -d "Yes the Mortality data set has been reported"
  "https://play.dhis2.org/demo/api/messageConversations/ZjHHSjyyeJ2"
  -H "Content-Type:text/plain" -u mobile:district -X POST
```

If all went according to plan you will receive a *200 OK* status code.

In 2.30 we added an URL search parameter:

```
queryString=?&queryOperator=?
```

The filter searches for matches in subject, text, and senders for message conversations. The default query operator is *token*, however other operators can be defined in the query.

Managing messages

As users receive and send messages, conversations will start to pile up in their inboxes, eventually becoming laborious to track. We will now have a look at managing a user's messages inbox by removing and marking conversations through the Web-API. We will do so by performing some maintenance in the inbox of the "DHIS Administrator" user.

First, let's have a look at removing a few messages from the inbox. Be sure to note that all removal operations described here only remove the relation between a user and a message conversation. In practical terms this means that we are not deleting the messages themselves (or any content for that matter) but are simply removing the message thread from the user such that it is no longer listed in the `/api/messageConversations` resource.

To remove a message conversation from a users inbox we need to issue a *DELETE* request to the resource identified by the id of the message conversation and the participating user. For example, to remove the user with id `xE7jOejl9FI` from the conversation with id `jMe43trzrdi`:

```
curl "https://play.dhis2.org/demo/api/33/messageConversations/jMe43trzrdi
```

If the request was successful the server will reply with a *200 OK*. The response body contains an XML or JSON object (according to the accept header of the request) containing the id of the removed user.

```
{
    "removed": ["xE7jOejl9FI"]
}
```

On failure the returned object will contain a message payload which describes the error.

```
{
    "message": "No user with uid: dMV6G0tPAEa"
}
```

The observant reader will already have noticed that the object returned on success in our example is actually a list of ids (containing a single entry). This is due to the endpoint also supporting batch removals. The request is made to the same *messageConversations* resource but follows slightly different semantics. For batch operations, the conversation ids are given as

query string parameters. The following example removes two separate message conversations for the current user:

```
curl "https://play.dhis2.org/demo/api/messageConversations?mc=WzMRrCosqc0&mc=lxCjiigqrJm"
  -X DELETE -u admin:district
```

If you have sufficient permissions, conversations can be removed on behalf of another user by giving an optional user id parameter.

```
curl "https://play.dhis2.org/demo/api/messageConversations?
mc=WzMRrCosqc0&mc=lxCjiigqrJm&user=PhzytPW3g2J"
  -X DELETE -u admin:district
```

As indicated, batch removals will return the same message format as for single operations. The list of removed objects will reflect successful removals performed. Partially erroneous requests (i.e. non-existing id) will therefore not cancel the entire batch operation.

Messages carry a boolean *read* property. This allows tracking whether a user has seen (opened) a message or not. In a typical application scenario (e.g. the DHIS2 web portal) a message will be marked read as soon as the user opens it for the first time. However, users might want to manage the read or unread status of their messages in order to keep track of certain conversations.

Marking messages read or unread follows similar semantics as batch removals, and also supports batch operations. To mark messages as read we issue a *POST* to the `messageConversations/read` resource with a request body containing one or more message ids. To mark messages as unread we issue an identical request to the `messageConversations/unread` resource. As is the case for removals, an optional *user* request parameter can be given.

Let's mark a couple of messages as read by the current user:

```
curl "https://play.dhis2.org/dev/api/messageConversations/read"
  -d '["ZrKML5WiyFm","Gc03smoTm6q"]' -X POST
  -H "Content-Type: application/json" -u admin:district
```

The response is a *200 OK* with the following JSON body:

```
{
    "markedRead": ["ZrKML5WiyFm", "Gc03smoTm6q"]
}
```

You can add recipients to an existing message conversation. The resource is located at:

```
/api/33/messageConversations/id/recipients
```

The options for this resource is a list of users, user groups and organisation units. The request should look like this:

```
{
    "users": [
        {
            "id": "OYLGMiazHtW"
```

```
        },
        {
            "id": "N3PZBUlN8vq"
        }
    ],
    "userGroups": [
        {
            "id": "DiszpKrYNg8"
        }
    ],
    "organisationUnits": [
        {
            "id": "DiszpKrYNg8"
        }
    ]
}
```

Message Attachments

Creating messages with attachments is done in two steps: uploading the file to the *attachments* resource, and then including one or several of the attachment IDs when creating a new message.

A POST request to the *attachments* resource will upload the file to the server.

```
curl -F file=@attachment.png "https://play.dhis2.org/demo/api/messageConversations/attachments"
  -u admin:district
```

The request returns an object that represents the attachment. The id of this object must be used when creating a message in order to link the attachment with the message.

```
{
    "created": "2018-07-20T16:54:18.210",
    "lastUpdated": "2018-07-20T16:54:18.212",
    "externalAccess": false,
    "publicAccess": "--------",
    "user": {
        "name": "John Traore",
        "created": "2013-04-18T17:15:08.407",
        "lastUpdated": "2018-03-09T23:06:54.512",
        "externalAccess": false,
        "displayName": "John Traore",
        "favorite": false,
        "id": "xE7jOejl9FI"
    },
    "lastUpdatedBy": {
        "id": "xE7jOejl9FI",
        "name": "John Traore"
    },
    "favorite": false,
    "id": "fTpI4GOmujz"
}
```

When creating a new message, the ids can be passed in the request body to link the uploaded files to the message being created.

```
{
    "subject": "Hey",
```

```
    "text": "How are you?",
    "users": [
        {
            "id": "OYLGMiazHtW"
        },
        {
            "id": "N3PZBUlN8vq"
        }
    ],
    "userGroups": [
        {
            "id": "ZoHNWQajIoe"
        }
    ],
    "organisationUnits": [
        {
            "id": "DiszpKrYNg8"
        }
    ],
    "attachments": ["fTpI4GOmujz", "h2ZsOxMFMfq"]
}
```

When replying to a message, the ids can be passed as a request parameter.

```
curl -d "Yes the Mortality data set has been reported"
  "https://play.dhis2.org/demo/api/33/messageConversations/ZjHHSjyyeJ2?
attachments=fTpI4GOmujz,h2ZsOxMFMfq"
  -H "Content-Type:text/plain" -u mobile:district -X POST
```

Once a message with an attachment has been created, the attached file can be accessed with a GET request to the following URL:

```
/api/messageConversations/<mcv-id>/<msg-id>/attachments/<attachment-id>
```

Where is the *message conversation* ID, is the ID of the *message* that contains the attachment and is the ID of the specific *message attachment*.

Tickets and Validation Result Notifications

You can use the "write feedback" tool to create tickets and messages. The only difference between a ticket and a message is that you can give a status and a priority to a ticket. To set the status:

```
POST /api/messageConversations/<uid>/status
```

To set the priority:

```
POST /api/messageConversations/<uid>/priority
```

In 2.29, messages generated by validation analysis now also be used in the status and priority properties. By default, messages generated by validation analysis will inherit the priority of the validation rule in question, or the highest importance if the message contains multiple rules.

In 2.30, validation rules can be assigned to any user while tickets still need to be assigned to a user in the system's feedback recipient group.

A list of valid status and priority values

Status	Priority
OPEN	LOW
PENDING	MEDIUM
INVALID	HIGH
SOLVED	

You can also add an internal message to a ticket, which can only be seen by users who have "Manage tickets" permissions. To create an internal reply, include the "internal" parameter, and set it to

```
curl -d "This is an internal message"
  "https://play.dhis2.org/demo/api/33/messageConversations/ZjHHSjyyeJ2?internal=true"
  -H "Content-Type:text/plain" -u admin:district -X POST
```

Interpretations

For resources related to data analysis in DHIS2, such as pivot tables, charts, maps, event reports and event charts, you can write and share data interpretations. An interpretation can be a comment, question, observation or interpretation about a data report or visualization.

```
/api/interpretations
```

Reading interpretations

To read interpretations we will interact with the `/api/interpretations` resource. A typical GET request using field filtering can look like this:

```
GET /api/interpretations?fields=*,comments[id,text,user,mentions]
```

The output in JSON response format could look like below (additional fields omitted for brevity):

```
{
    "interpretations": [
        {
            "id": "XSHiFlHAhhh",
            "created": "2013-05-30T10:24:06.181+0000",
            "text": "Data looks suspicious, could be a data entry mistake.",
            "type": "REPORT_TABLE",
            "likes": 2,
            "user": {
                "id": "uk7diLujYif"
            },
            "reportTable": {
                "id": "LcSxnfeBxyi"
            },
            "visualization": {
                "id": "LcSxnfeBxyi"
```

```
            }
        },
        {
            "id": "kr4AnZmYL43",
            "created": "2013-05-29T14:47:13.081+0000",
            "text": "Delivery rates in Bo looks high.",
            "type": "CHART",
            "likes": 3,
            "user": {
                "id": "uk7diLujYif"
            },
            "chart": {
                "id": "HDEDqV3yv3H"
            },
            "visualization": {
                "id": "HDEDqV3yv3H"
            },
            "mentions": [
                {
                    "created": "2018-06-25T10:25:54.498",
                    "username": "boateng"
                }
            ],
            "comments": [
                {
                    "id": "iB4Etq8yTE6",
                    "text": "This report indicates a surge.",
                    "user": {
                        "id": "B4XIfwOcGyI"
                    }
                },
                {
                    "id": "iB4Etq8yTE6",
                    "text": "Likely caused by heavy rainfall.",
                    "user": {
                        "id": "B4XIfwOcGyI"
                    }
                },
                {
                    "id": "SIjkdENan8p",
                    "text": "Have a look at this @boateng.",
                    "user": {
                        "id": "xE7jOejl9FI"
                    },
                    "mentions": [
                        {
                            "created": "2018-06-25T10:03:52.316",
                            "username": "boateng"
                        }
                    ]
                }
            ]
        }
    ]
}
```


Interpretation fields

Field	Description
id	The interpretation identifier.
created	The time of when the interpretation was created.

Field	Description
type	The type of analytical object being interpreted. Valid options: REPORT_TABLE, CHART, MAP, EVENT_REPORT, EVENT_CHART, DATASET_REPORT.
user	Association to the user who created the interpretation.
reportTable	Association to the report table if type is REPORT_TABLE.
chart	Association to the chart if type is CHART.
visualization	Association to the visualization if type is CHART or REPORT_TABLE (**both types are in deprecation process in favour of VISUALIZATION**).
map	Association to the map if type is MAP.
eventReport	Association to the event report is type is EVENT_REPORT.
eventChart	Association to the event chart if type is EVENT_CHART.
dataSet	Association to the data set if type is DATASET_REPORT.
comments	Array of comments for the interpretation. The text field holds the actual comment.
mentions	Array of mentions for the interpretation. A list of users identifiers.

For all analytical objects you can append */data* to the URL to retrieve the data associated with the resource (as opposed to the metadata). As an example, by following the map link and appending /data one can retrieve a PNG (image) representation of the thematic map through the following URL:

```
https://play.dhis2.org/demo/api/maps/bhmHJ4ZCdCd/data
```

For all analytical objects you can filter by *mentions*. To retrieve all the interpretations/comments where a user has been mentioned you have three options. You can filter by the interpretation mentions (mentions in the interpretation description):

```
GET /api/interpretations?fields=*,comments[*]&filter=mentions.username:in:[boateng]
```

You can filter by the interpretation comments mentions (mentions in any comment):

```
GET /api/interpretations?fields=*,comments[*]
  &filter=comments.mentions.username:in:[boateng]
```

You can filter by intepretations which contains the mentions either in the interpretation or in any comment (OR junction):

```
GET /api/interpretations?fields=*,comments[*]&filter=mentions:in:[boateng]
```

Writing interpretations

When writing interpretations you will supply the interpretation text as the request body using a POST request with content type "text/plain". The URL pattern looks like the below, where {object-type} refers to the type of the object being interpreted and {object-id} refers to the identifier of the object being interpreted.

```
/api/interpretations/{object-type}/{object-id}
```

Valid options for object type are *reportTable*, *chart*, *map*, *eventReport*, *eventChart* and *dataSetReport*.

Some valid examples for interpretations are listed below.

> **Note**
>
> The `charts` and `reportTables` APIs are deprecated. We recommend using the `visualizations` API instead.

```
/api/interpretations/reportTable/yC86zJxU1i1
/api/interpretations/chart/ZMuYVhtIceD
/api/interpretations/visualization/hQxZGXqnLS9
/api/interpretations/map/FwLHSMCejFu
/api/interpretations/eventReport/xJmPLGP3Cde
/api/interpretations/eventChart/nEzXB2M9YBz
/api/interpretations/dataSetReport/tL7eCjmDIgM
```

As an example, we will start by writing an interpretation for the chart with identifier *EbRN2VIbPdV*. To write chart interpretations we will interact with the `/api/interpretations/chart/{chartId}` resource. The interpretation will be the request body. Based on this we can put together the following request using cURL:

```
curl -d "This chart shows a significant ANC 1-3 dropout" -X POST
  "https://play.dhis2.org/demo/api/interpretations/chart/EbRN2VIbPdV"
  -H "Content-Type:text/plain" -u admin:district
```

Notice that the response provides a Location header with a value indicating the location of the created interpretation. This is useful from a client perspective when you would like to add a comment to the interpretation.

Updating and removing interpretations

To update an existing interpretation you can use a PUT request where the interpretation text is the request body using the following URL pattern, where {id} refers to the interpretation identifier:

```
/api/interpretations/{id}
```

Based on this we can use curl to update the interpretation:

```
curl -d "This charts shows a high dropout" -X PUT
  "https://play.dhis2.org/demo/api/interpretations/chart/EV08iI1cJRA"
  -H "Content-Type:text/plain" -u admin:district
```

You can use the same URL pattern as above using a DELETE request to remove the interpretation.

Creating interpretation comments

When writing comments to interpretations you will supply the comment text as the request body using a POST request with content type "text/plain". The URL pattern looks like the below, where {interpretation-id} refers to the interpretation identifier.

```
/api/interpretations/{interpretation-id}/comments
```

Second, we will write a comment to the interpretation we wrote in the example above. By looking at the interpretation response you will see that a *Location* header is returned. This header tells us the URL of the newly created interpretation and from that, we can read its identifier. This identifier is randomly generated so you will have to replace the one in the command below with your own. To write a comment we can interact with the `/api/interpretations/{id}/comments` resource like this:

```
curl -d "An intervention is needed" -X POST
  "https://play.dhis2.org/demo/api/interpretations/j8sjHLkK8uY/comments"
  -H "Content-Type:text/plain" -u admin:district
```

Updating and removing interpretation comments

To updating an interpretation comment you can use a PUT request where the comment text is the request body using the following URL pattern:

```
/api/interpretations/{interpretation-id}/comments/{comment-id}
```

Based on this we can use curl to update the comment:

```
curl "https://play.dhis2.org/demo/api/interpretations/j8sjHLkK8uY/comments/idAzzhVWvh2"
  -d "I agree with that." -X PUT -H "Content-Type:text/plain" -u admin:district
```

You can use the same URL pattern as above using a DELETE request to the remove the interpretation comment.

Liking interpretations

To like an interpretation you can use an empty POST request to the *like* resource:

```
POST /api/interpretations/{id}/like
```

A like will be added for the currently authenticated user. A user can only like an interpretation once.

To remove a like for an interpretation you can use a DELETE request to the same resource as for the like operation.

The like status of an interpretation can be viewed by looking at the regular Web API representation:

```
GET /api/interpretations/{id}
```

The like information is found in the *likes* field, which represents the number of likes, and the *likedBy* array, which enumerates the users who have liked the interpretation.

```
{
    "id": "XSHiFlHAhhh",
    "text": "Data looks suspicious, could be a data entry mistake.",
    "type": "REPORT_TABLE",
    "likes": 2,
    "likedBy": [
        {
            "id": "k7Hg12fJ2f1"
        },
        {
            "id": "gYhf26fFkjFS"
        }
    ]
}
```

Viewing analytical resource representations

DHIS2 has several resources for data analysis. These resources include *charts*, *maps*, *reportTables*, *reports* and *documents*. By visiting these resources you will retrieve information about the resource. For instance, by navigating to `/api/charts/R0DVGvXDUNP` the response will contain the name, last date of modification and so on for the chart. To retrieve the analytical representation, for instance, a PNG representation of the chart, you can append */data* to all these resources. For instance, by visiting `/api/charts/R0DVGvXDUNP/data` the system will return a PNG image of the chart.

Analytical resources

Resource	Description	Data URL	Resource representations
charts	Charts	/api/charts/<identifier>/data	png
eventCharts	Event charts	/api/eventCharts/<identifier>/data	png
maps	Maps	/api/maps/<identifier>/data	png
reportTables	Pivot tables	/api/reportTables/<identifier>/data	json | jsonp | html | xml | pdf | xls | csv
reports	Standard reports	/api/reports/<identifier>/data	pdf | xls | html
documents	Resources	/api/documents/<identifier>/data	<follows document>

The data content of the analytical representations can be modified by providing a *date* query parameter. This requires that the analytical resource is set up for relative periods for the period dimension.

Data query parameters

Query parameter	Value	Description
date	Date in yyyy-MM-dd format	Basis for relative periods in report (requires relative periods)

Query parameters for png / image types (charts, maps)

Query parameter	Description
width	Width of image in pixels
height	Height of image in pixels

Some examples of valid URLs for retrieving various analytical representations are listed below.

```
/api/charts/R0DVGvXDUNP/data
/api/charts/R0DVGvXDUNP/data?date=2013-06-01

/api/reportTables/jIISuEWxmoI/data.html
/api/reportTables/jIISuEWxmoI/data.html?date=2013-01-01
/api/reportTables/FPmvWs7bn2P/data.xls
/api/reportTables/FPmvWs7bn2P/data.pdf

/api/maps/DHE98Gsynpr/data
/api/maps/DHE98Gsynpr/data?date=2013-07-01

/api/reports/OeJsA6K1Otx/data.pdf
/api/reports/OeJsA6K1Otx/data.pdf?date=2014-01-01
```

Plugins

DHIS2 comes with plugins which enable you to embed live data directly in your web portal or web site. Currently, plugins exist for charts, maps and pivot tables.

Please be aware that all of the code examples in this section are for demonstration purposes only. They should not be used as is in production systems. To make things simple, the credentials (admin/district) have been embedded into the scripts. In a real scenario, you should never expose credentials in javascript as it opens a vulnerability to the application. In addition, you would create a user with more minimal privileges rather than make use of a superuser to fetch resources for your portal.

It is possible to workaround exposing the credentials by using a reverse proxy such as nginx or apache2. The proxy can be configured to inject the required Authorization header for only the endpoints that you wish to make public. There is some documentation to get you started in the section of the implementers manual which describes reverse proxy configuration.

Embedding pivot tables with the Pivot Table plug-in

In this example, we will see how we can embed good-looking, light-weight html pivot tables with data served from a DHIS2 back-end into a Web page. To accomplish this we will use the Pivot table plug-in. The plug-in is written in Javascript and depends on the jQuery library only. A complete working example can be found at http://play.dhis2.org/portal/table.html. Open the page in a web browser and view the source to see how it is set up.

We start by having a look at what the complete html file could look like. This setup puts two tables in our web page. The first one is referring to an existing table. The second is configured inline.

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://dhis2-cdn.org/v227/plugin/jquery-2.2.4.min.js"></script>
        <script src="https://dhis2-cdn.org/v227/plugin/reporttable.js"></script>

        <script>
            reportTablePlugin.url = "https://play.dhis2.org/demo";
```

```
            reportTablePlugin.username = "admin";
            reportTablePlugin.password = "district";
            reportTablePlugin.loadingIndicator = true;

            // Referring to an existing table through the id parameter, render to "report1" div

            var r1 = { el: "report1", id: "R0DVGvXDUNP" };

            // Table configuration, render to "report2" div

            var r2 = {
                el: "report2",
                columns: [
                    {
                        dimension: "dx",
                        items: [{ id: "YtbsuPPo010" }, { id: "l6byfWFUGaP" }],
                    },
                ],
                rows: [{ dimension: "pe", items: [{ id: "LAST_12_MONTHS" }] }],
                filters: [{ dimension: "ou", items: [{ id: "USER_ORGUNIT" }] }],

                // All following properties are optional
                title: "My custom title",
                showColTotals: false,
                showRowTotals: false,
                showColSubTotals: false,
                showRowSubTotals: false,
                showDimensionLabels: false,
                hideEmptyRows: true,
                skipRounding: true,
                aggregationType: "AVERAGE",
                showHierarchy: true,
                completedOnly: true,
                displayDensity: "COMFORTABLE",
                fontSize: "SMALL",
                digitGroupSeparator: "COMMA",
                legendSet: { id: "fqs276KXCXi" },
            };

            reportTablePlugin.load([r1, r2]);
        </script>
    </head>

    <body>
        <div id="report1"></div>
        <div id="report2"></div>
    </body>
</html>
```

Two files are included in the header section of the HTML document. The first file is the jQuery JavaScript library (we use the DHIS2 content delivery network in this case). The second file is the Pivot table plug-in. Make sure the path is pointing to your DHIS2 server installation.

Now let us have a look at the various options for the Pivot tables. One property is required: *el* (please refer to the table below). Now, if you want to refer to pre-defined tables already made inside DHIS2 it is sufficient to provide the additional *id* parameter. If you instead want to configure a pivot table dynamically you should omit the id parameter and provide data dimensions inside a *columns* array, a *rows* array and optionally a *filters* array instead.

A data dimension is defined as an object with a text property called *dimension*. This property accepts the following values: *dx* (indicator, data element, data element operand, data set, event

data item and program indicator), *pe* (period), *ou* (organisation unit) or the id of any organisation unit group set or data element group set (can be found in the web api). The data dimension also has an array property called *items* which accepts objects with an *id* property.

To sum up, if you want to have e.g. "ANC 1 Coverage", "ANC 2 Coverage" and "ANC 3 Coverage" on the columns in your table you can make the following *columns* config:

```
columns: [{
  dimension: "dx",
  items: [
    {id: "Uvn6LCg7dVU"}, // the id of ANC 1 Coverage
    {id: "OdiHJayrsKo"}, // the id of ANC 2 Coverage
    {id: "sB79w2hiLp8"}  // the id of ANC 3 Coverage
  ]
}]
```

Pivot table plug-in configuration

Param	Type	Required	Options (default first)	Description
url	string	Yes		Base URL of the DHIS2 server
username	string	Yes (if cross-domain)		Used for authentication if the server is running on a different domain
password	string	Yes (if cross-domain)		Used for authentication if the server is running on a different domain
loadingIndicator	boolean	No		Whether to show a loading indicator before the table appears

Pivot table configuration

Param	Type	Required	Options (default first)	Description
el	string	Yes		Identifier of the HTML element to render the table in your web page
id	string	No		Identifier of a pre-defined table (favorite) in DHIS2
columns	array	Yes (if no id provided)		Data dimensions to include in table as columns

Param	Type	Required	Options (default first)	Description
rows	array	Yes (if no id provided)		Data dimensions to include in table as rows
filter	array	No		Data dimensions to include in table as filters
title	string	No		Show a custom title above the table
showColTotals	boolean	No	true | false	Whether to display totals for columns
showRowTotals	boolean	No	true | false	Whether to display totals for rows
showColSubTotals	boolean	No	true | false	Whether to display sub-totals for columns
showRowSubTotals	boolean	No	true | false	Whether to display sub-totals for rows
showDimensionLabels	boolean	No	true | false	Whether to display the name of the dimension top-left in the table
hideEmptyRows	boolean	No	false | true	Whether to hide rows with no data
skipRounding	boolean	No	false | true	Whether to skip rounding of data values
completedOnly	boolean	No	false | true	Whether to only show completed events
showHierarchy	boolean	No	false | true	Whether to extend orgunit names with the name of all anchestors

Param	Type	Required	Options (default first)	Description
aggregationType	string	No	"SUM" |"AVERAGE" | "AVERAGE_SUM_ORG_UNIT"|"LAST"|"LAST_AVERAGE_ORG_UNIT"| "COUNT" | "STDDEV" | "VARIANCE" | "MIN" | "MAX"	Override the data element's default aggregation type
displayDensity	string	No	"NORMAL" | "COMFORTABLE" | "COMPACT"	The amount of space inside table cells
fontSize	string	No	"NORMAL" | "LARGE" | "SMALL"	Table font size
digitGroupSeparator	string	No	"SPACE" | "COMMA" | "NONE"	How values are formatted: 1 000 | 1,000 | 1000
legendSet	object	No		Color the values in the table according to the legend set
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with curretn user, single or array
relativePeriodDate	string	No		Date identifier e.g: "2016-01-01". Overrides the start date of the relative period

Embedding charts with the Visualizer chart plug-in

In this example, we will see how we can embed good-looking Highcharts charts (http://www.highcharts.com) with data served from a DHIS2 back-end into a Web page. To accomplish this we will use the DHIS2 Visualizer plug-in. The plug-in is written in JavaScript and depends on the jQuery library. A complete working example can be found at http://play.dhis2.org/portal/chart.html. Open the page in a web browser and view the source to see how it is set up.

We start by having a look at what the complete html file could look like. This setup puts two charts on our web page. The first one is referring to an existing chart. The second is configured inline.

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <script src="https://dhis2-cdn.org/v227/plugin/jquery-2.2.4.min.js"></script>
        <script src="https://dhis2-cdn.org/v227/plugin/chart.js"></script>

        <script>
            chartPlugin.url = "https://play.dhis2.org/demo";
            chartPlugin.username = "admin";
            chartPlugin.password = "district";
            chartPlugin.loadingIndicator = true;

            // Referring to an existing chart through the id parameter, render to "report1" div

            var r1 = { el: "report1", id: "R0DVGvXDUNP" };

            // Chart configuration, render to "report2" div

            var r2 = {
                el: "report2",
                columns: [
                    {
                        dimension: "dx",
                        items: [{ id: "YtbsuPPo010" }, { id: "l6byfWFUGaP" }],
                    },
                ],
                rows: [{ dimension: "pe", items: [{ id: "LAST_12_MONTHS" }] }],
                filters: [{ dimension: "ou", items: [{ id: "USER_ORGUNIT" }] }],

                // All following properties are optional
                title: "Custom title",
                type: "line",
                showValues: false,
                hideEmptyRows: true,
                regressionType: "LINEAR",
                completedOnly: true,
                targetLineValue: 100,
                targetLineTitle: "My target line title",
                baseLineValue: 20,
                baseLineTitle: "My base line title",
                aggregationType: "AVERAGE",
                rangeAxisMaxValue: 100,
                rangeAxisMinValue: 20,
                rangeAxisSteps: 5,
                rangeAxisDecimals: 2,
                rangeAxisTitle: "My range axis title",
                domainAxisTitle: "My domain axis title",
                hideLegend: true,
            };

            // Render the charts

            chartPlugin.load(r1, r2);
        </script>
    </head>

    <body>
        <div id="report1"></div>
        <div id="report2"></div>
    </body>
</html>
```

Two files are included in the header section of the HTML document. The first file is the jQuery JavaScript library (we use the DHIS2 content delivery network in this case). The second file is the Visualizer chart plug-in. Make sure the path is pointing to your DHIS2 server installation.

Now let us have a look at the various options for the charts. One property is required: *el* (please refer to the table below). Now, if you want to refer to pre-defined charts already made inside DHIS2 it is sufficient to provide the additional *id* parameter. If you instead want to configure a chart dynamically you should omit the id parameter and provide data dimensions inside a *columns* array, a *rows* array and optionally a *filters* array instead.

A data dimension is defined as an object with a text property called *dimension*. This property accepts the following values: *dx* (indicator, data element, data element operand, data set, event data item and program indicator), *pe* (period), *ou* (organisation unit) or the id of any organisation unit group set or data element group set (can be found in the web api). The data dimension also has an array property called *items* which accepts objects with an *id* property.

To sum up, if you want to have e.g. "ANC 1 Coverage", "ANC 2 Coverage" and "ANC 3 Coverage" on the columns in your chart you can make the following *columns* config:

```
columns: [{
  dimension: "dx",
  items: [
    {id: "Uvn6LCg7dVU"}, // the id of ANC 1 Coverage
    {id: "OdiHJayrsKo"}, // the id of ANC 2 Coverage
    {id: "sB79w2hiLp8"}  // the id of ANC 3 Coverage
  ]
}]
```

Chart plug-in configuration

Param	Type	Required	Options (default first)	Description
url	string	Yes		Base URL of the DHIS2 server
username	string	Yes (if cross-domain)		Used for authentication if the server is running on a different domain
password	string	Yes (if cross-domain)		Used for authentication if the server is running on a different domain
loadingIndicator	boolean	No		Whether to show a loading indicator before the chart appears

Chart configuration

Param	Type	Required	Options (default first)	Description
el	string	Yes		Identifier of the HTML element to render the chart in your web page

Param	Type	Required	Options (default first)	Description
id	string	No		Identifier of a pre-defined chart (favorite) in DHIS
type	string	No	column | stackedcolumn | bar | stackedbar | line | area | pie | radar | gauge	Chart type
columns	array	Yes (if no id provided)		Data dimensions to include in chart as series
rows	array	Yes (if no id provided)		Data dimensions to include in chart as category
filter	array	No		Data dimensions to include in chart as filters
title	string	No		Show a custom title above the chart
showValues	boolean	No	false | true	Whether to display data values on the chart
hideEmptyRows	boolean	No	false | true	Whether to hide empty categories
completedOnly	boolean	No	false | true	Whether to only show completed events
regressionType	string	No	"NONE" | "LINEAR"	Show trend lines
targetLineValue	number	No		Display a target line with this value
targetLineTitle	string	No		Display a title on the target line (does not apply without a target line value)
baseLineValue	number	No		Display a base line with this value
baseLineTitle	string	No		Display a title on the base line (does not apply without a base line value)

Param	Type	Required	Options (default first)	Description
rangeAxisTitle	number	No		Title to be displayed along the range axis
rangeAxisMaxValue	number	No		Max value for the range axis to display
rangeAxisMinValue	number	No		Min value for the range axis to display
rangeAxisSteps	number	No		Number of steps for the range axis to display
rangeAxisDecimals	number	No		Bumber of decimals for the range axis to display
domainAxisTitle	number	No		Title to be displayed along the domain axis
aggregationType	string	No	"SUM" |"AVERAGE" | "AVERAGE_SUM_ORG_UNIT"|"LAST"|"LAST_AVERAGE_ORG_UNIT"| "COUNT" | "STDDEV" | "VARIANCE" | "MIN" | "MAX"	Override the data element's default aggregation type
hideLegend	boolean	No	false | true	Whether to hide the series legend
hideTitle	boolean	No	false | true	Whether to hide the chart title
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with curretn user, single or array
relativePeriodDate	string	No		Date identifier e.g: "2016-01-01". Overrides the start date of the relative period

Embedding maps with the GIS map plug-in

In this example we will see how we can embed maps with data served from a DHIS2 back-end into a Web page. To accomplish this we will use the GIS map plug-in. The plug-in is written in JavaScript and depends on the Ext JS library only. A complete working example can be found at http://play.dhis2.org/portal/map.html. Open the page in a web browser and view the source to see how it is set up.

We start by having a look at what the complete html file could look like. This setup puts two maps on our web page. The first one is referring to an existing map. The second is configured inline.

```
<!DOCTYPE html>
<html>
    <head>
        <link
            rel="stylesheet"
            type="text/css"
            href="http://dhis2-cdn.org/v215/ext/resources/css/ext-plugin-gray.css"
        />
        <script src="http://dhis2-cdn.org/v215/ext/ext-all.js"></script>
        <script src="https://maps.google.com/maps/api/js?sensor=false"></script>
        <script src="http://dhis2-cdn.org/v215/openlayers/OpenLayers.js"></script>
        <script src="http://dhis2-cdn.org/v215/plugin/map.js"></script>

        <script>
            var base = "https://play.dhis2.org/demo";

            // Login - if OK, call the setLinks function

            Ext.onReady(function () {
                Ext.Ajax.request({
                    url: base + "dhis-web-commons-security/login.action",
                    method: "POST",
                    params: { j_username: "portal", j_password: "Portal123" },
                    success: setLinks,
                });
            });

            function setLinks() {
                DHIS.getMap({ url: base, el: "map1", id: "ytkZY3ChM6J" });

                DHIS.getMap({
                    url: base,
                    el: "map2",
                    mapViews: [
                        {
                            columns: [
                                {
                                    dimension: "in",
                                    items: [{ id: "Uvn6LCg7dVU" }],
                                },
                            ], // data
                            rows: [
                                {
                                    dimension: "ou",
                                    items: [
                                        { id: "LEVEL-3" },
                                        { id: "ImspTQPwCqd" },
                                    ],
                                },
                            ], // organisation units,
```

```
                            filters: [
                                {
                                    dimension: "pe",
                                    items: [{ id: "LAST_3_MONTHS" }],
                                },
                            ], // period
                            // All following options are optional
                            classes: 7,
                            colorLow: "02079c",
                            colorHigh: "e5ecff",
                            opacity: 0.9,
                            legendSet: { id: "fqs276KXCXi" },
                        },
                    ],
                });
            }
        </script>
    </head>

    <body>
        <div id="map1"></div>
        <div id="map2"></div>
    </body>
</html>
```

Four files and Google Maps are included in the header section of the HTML document. The first two files are the Ext JS JavaScript library (we use the DHIS2 content delivery network in this case) and its stylesheet. The third file is the OpenLayers JavaScript mapping framework (http://openlayers.org) and finally we include the GIS map plug-in. Make sure the path is pointing to your DHIS2 server installation.

```
<link rel="stylesheet" type="text/css" href="http://dhis2-cdn.org/v215/ext/resources/css/ext-
plugin-gray.css" />
<script src="http://dhis2-cdn.org/v215/ext/ext-all.js"></script>
<script src="https://maps.google.com/maps/api/js?sensor=false"></script>
<script src="http://dhis2-cdn.org/v215/openlayers/OpenLayers.js"></script>
<script src="http://dhis2-cdn.org/v215/plugin/map.js"></script>
```

To authenticate with the DHIS2 server we use the same approach as in the previous section. In the header of the HTML document we include the following Javascript inside a script element. The *setLinks* method will be implemented later. Make sure the *base* variable is pointing to your DHIS2 installation.

```
Ext.onReady( function() {
  Ext.Ajax.request({
    url: base + "dhis-web-commons-security/login.action",
    method: "POST",
    params: { j_username: "portal", j_password: "Portal123" },
    success: setLinks
  });
});
```

Now let us have a look at the various options for the GIS plug-in. Two properties are required: *el* and *url* (please refer to the table below). Now, if you want to refer to pre-defined maps already made in the DHIS2 GIS it is sufficient to provide the additional *id* parameter. If you instead want to configure a map dynamically you should omit the id parameter and provide *mapViews* (layers)

instead. They should be configured with data dimensions inside a *columns* array, a *rows* array and optionally a *filters* array instead.

A data dimension is defined as an object with a text property called *dimension*. This property accepts the following values: *in* (indicator), *de* (data element), *ds* (data set), *dc* (data element operand), *pe* (period), *ou* (organisation unit) or the id of any organisation unit group set or data element group set (can be found in the web api). The data dimension also has an array property called *items* which accepts objects with an *id* property.

To sum up, if you want to have a layer with e.g. "ANC 1 Coverage" in your map you can make the following *columns* config:

```
columns: [{
  dimension: "in", // could be "in", "de", "ds", "dc", "pe", "ou" or any dimension id
  items: [{id: "Uvn6LCg7dVU"}], // the id of ANC 1 Coverage
}]
```

GIS map plug-in configuration

Param	Type	Required	Options (default first)	Description
el	string	Yes		Identifier of the HTML element to render the map in your web page
url	string	Yes		Base URL of the DHIS2 server
id	string	No		Identifier of a pre-defined map (favorite) in DHIS
baseLayer	string/boolean	No	'gs', 'googlestreets' | 'gh', 'googlehybrid' | 'osm', 'openstreetmap' | false, null, 'none', 'off'	Show background map
hideLegend	boolean	No	false | true	Hide legend panel
mapViews	array	Yes (if no id provided)		Array of layers

If no id is provided you must add map view objects with the following config options:

Map plug-in configuration

layer	string	No	"thematic1" | "thematic2" | "thematic3" | "thematic4" | "boundary" | "facility" |	The layer to which the map view content should be added
columns	array	Yes		Indicator, data element, data operand or data set (only one will be used)
rows	array	Yes		Organisation units (multiple allowed)
filter	array	Yes		Period (only one will be used)
classes	integer	No	5 | 1-7	The number of automatic legend classes
method	integer	No	2 | 3	Legend calculation method where 2 = equal intervals and 3 = equal counts
colorLow	string	No	"ff0000" (red) | Any hex color	The color representing the first automatic legend class
colorHigh	string	No	"00ff00" (green) | Any hex color	The color representing the last automatic legend class
radiusLow	integer	No	5 | Any integer	Only applies for facilities (points) - radius of the point with lowest value
radiusHigh	integer	No	15 | Any integer	Only applies for facilities (points) - radius of the point with highest value
opacity	double	No	0.8 | 0 - 1	Opacity/ transparency of the layer content
legendSet	object	No		Pre-defined legend set. Will override the automatic legend set.

labels	boolean/object	No	false | true | object properties: fontSize (integer), color (hex string), strong (boolean), italic (boolean)	Show labels on the map
width	integer	No		Width of map
height	integer	No		Height of map
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with current user, single or array

We continue by adding one pre-defined and one dynamically configured map to our HTML document. You can browse the list of available maps using the Web API here: http://play.dhis2.org/demo/api/33/maps.

```
function setLinks() {
  DHIS.getMap({ url: base, el: "map1", id: "ytkZY3ChM6J" });

  DHIS.getMap({
 url: base,
 el: "map2",
 mapViews: [
   columns: [ // Chart series
  columns: [{dimension: "in", items: [{id: "Uvn6LCg7dVU"}]}], // data
   ],
   rows: [ // Chart categories
  rows: [{dimension: "ou", items: [{id: "LEVEL-3"}, {id: "ImspTQPwCqd"}]}], // organisation units
   ],
   filters: [
  filters: [{dimension: "pe", items: [{id: "LAST_3_MONTHS"}]}], // period
   ],
   // All following options are optional
   classes: 7,
   colorLow: "02079c",
   colorHigh: "e5ecff",
   opacity: 0.9,
   legendSet: {id: "fqs276KXCXi"}
 ]
  });
}
```

Finally we include some *div* elements in the body section of the HTML document with the identifiers referred to in the plug-in JavaScript.

```
<div id="map1"></div>
<div id="map2"></div>
```

To see a complete working example please visit http://play.dhis2.org/portal/map.html.

SQL views

The SQL views resource allows you to create and retrieve the result set of SQL views. The SQL views can be executed directly against the database and render the result set through the Web API resource.

```
/api/sqlViews
```

SQL views are useful for creating data views which may be more easily constructed with SQL compared combining the multiple objects of the Web API. As an example, lets assume we have been asked to provide a view of all organization units with their names, parent names, organization unit level and name, and the coordinates listed in the database. The view might look something like this:

```
SELECT ou.name as orgunit, par.name as parent, ou.coordinates, ous.level, oul.name from
organisationunit ou
INNER JOIN _orgunitstructure ous ON ou.organisationunitid = ous.organisationunitid
INNER JOIN organisationunit par ON ou.parentid = par.organisationunitid
INNER JOIN orgunitlevel oul ON ous.level = oul.level
WHERE ou.coordinates is not null
ORDER BY oul.level, par.name, ou.name
```

We will use *curl* to first execute the view on the DHIS2 server. This is essentially a materialization process, and ensures that we have the most recent data available through the SQL view when it is retrieved from the server. You can first look up the SQL view from the api/sqlViews resource, then POST using the following command:

```
curl "https://play.dhis2.org/demo/api/sqlViews/dI68mLkP1wN/execute" -X POST -u admin:district
```

The next step in the process is the retrieval of the data.The basic structure of the URL is as follows

```
http://{server}/api/sqlViews/{id}/data(.csv)
```

The `{server}` parameter should be replaced with your own server. The next part of the URL `/api/sqlViews/` should be appended with the specific SQL view identifier. Append either `data` for XML data or `data.csv` for comma delimited values. Support response formats are json, xml, csv, xls, html and html+css. As an example, the following command would retrieve XML data for the SQL view defined above.

```
curl "https://play.dhis2.org/demo/api/sqlViews/dI68mLkP1wN/data.csv" -u admin:district
```

There are three types of SQL views:

- *SQL view:* Standard SQL views.
- *Materialized SQL view:* SQL views which are materialized, meaning written to disk. Needs to be updated to reflect changes in underlying tables. Supports criteria to filter result set.
- *SQL queries:* Plain SQL queries. Support inline variables for customized queries.

Criteria

You can do simple filtering on the columns in the result set by appending *criteria* query parameters to the URL, using the column names and filter values separated by columns as parameter values, on the following format:

```
/api/sqlViews/{id}/data?criteria=col1:value1&criteria=col2:value2
```

As an example, to filter the SQL view result set above to only return organisation units at level 4 you can use the following URL:

```
https://play.dhis2.org/demo/api/sqlViews/dI68mLkP1wN/data.csv?criteria=level:4
```

Variables

SQL views support variable substitution. Variable substitution is only available for SQL view of type *query*, meaning SQL views which are not created in the database but simply executed as regular SQL queries. Variables can be inserted directly into the SQL query and must be on this format:

```
${variable-key}
```

As an example, an SQL query that retrieves all data elements of a given value type where the value type is defined through a variable can look like this:

```
select * from dataelement where valuetype = '${valueType}';
```

These variables can then be supplied as part of the URL when requested through the *sqlViews* Web API resource. Variables can be supplied on the following format:

```
/api/sqlViews/{id}/data?var=key1:value1&var=key2:value2
```

An example query corresponding to the example above can look like this:

```
/api/sqlViews/dI68mLkP1wN/data.json?var=valueType:int
```

The *valueType* variable will be substituted with the *int* value, and the query will return data elements with int value type.

The variable parameter must contain alphanumeric characters only. The variables must contain alphanumeric, dash, underscore and whitespace characters only.

SQL Views of type *query* also support two system-defined variables that allow the query to access information about the user executing the view:

variable	means
${_current_user_id}	the user's database id
${_current_username}	the user's username

Values for these variables cannot be supplied as part of the URL. They are always filled with information about the user.

For example, the following SQL view of type *query* shows all the organisation units that are assigned to the user:

```
select ou.path, ou.name
from organisationunit ou_user
join organisationunit ou on ou.path like ou_user.path || '%'
join usermembership um on um.organisationunitid = ou_user.organisationunitid
where um.userinfoid = ${_current_user_id}
order by ou.path
```

Filtering

The SQL view api supports data filtering, equal to the metadata object filter. For a complete list of filter operators you can look at the documentation for metadata object filter.

To use filters, simply add them as parameters at the end of the request url for your SQL view like this:

```
/api/sqlViews/w3UxFykyHFy/data.json?filter=orgunit_level:eq:2&filter=orgunit_name:ilike:bo
```

This request will return a result including org units with "bo" in the name and which has org unit level 2.

The following example will return all org units with `orgunit_level` 2 or 4:

```
/api/sqlViews/w3UxFykyHFy/data.json?filter=orgunit_level:in:[2,4]
```

And last, an example to return all org units that does not start with "Bo"

```
/api/sqlViews/w3UxFykyHFy/data.json?filter=orgunit_name:!like:Bo
```

Dashboard

The dashboard is designed to give you an overview of multiple analytical items like maps, charts, pivot tables and reports which together can provide a comprehensive overview of your data. Dashboards are available in the Web API through the *dashboards* resource. A dashboard contains a list of dashboard *items*. An item can represent a single resource, like a chart, map or report table, or represent a list of links to analytical resources, like reports, resources, tabular reports and users. A dashboard item can contain up to eight links. Typically, a dashboard client could choose to visualize the single-object items directly in a user interface, while rendering the multi-object items as clickable links.

```
/api/dashboards
```

Browsing dashboards

To get a list of your dashboards with basic information including identifier, name and link in JSON format you can make a *GET* request to the following URL:

```
/api/dashboards.json
```

The dashboards resource will provide a list of dashboards. Remember that the dashboard object is shared so the list will be affected by the currently authenticated user. You can retrieve more information about a specific dashboard by following its link, similar to this:

```
/api/dashboards/vQFhmLJU5sK.json
```

A dashboard contains information like name and creation date and an array of dashboard items. The response in JSON format will look similar to this response (certain information has been removed for the sake of brevity).


```
{
    "lastUpdated": "2013-10-15T18:17:34.084+0000",
    "id": "vQFhmLJU5sK",
    "created": "2013-09-08T20:55:58.060+0000",
    "name": "Mother and Child Health",
    "href": "https://play.dhis2.org/demo/api/dashboards/vQFhmLJU5sK",
    "publicAccess": "--------",
    "externalAccess": false,
    "itemCount": 17,
    "displayName": "Mother and Child Health",
    "access": {
        "update": true,
        "externalize": true,
        "delete": true,
        "write": true,
        "read": true,
        "manage": true
    },
    "user": {
        "id": "xE7jOejl9FI",
        "name": "John Traore",
        "created": "2013-04-18T15:15:08.407+0000",
        "lastUpdated": "2014-12-05T03:50:04.148+0000",
        "href": "https://play.dhis2.org/demo/api/users/xE7jOejl9FI"
    },
    "dashboardItems": [
        {
            "id": "bu1IAnPFa9H",
            "created": "2013-09-09T12:12:58.095+0000",
            "lastUpdated": "2013-09-09T12:12:58.095+0000"
        },
        {
            "id": "ppFEJmWWDa1",
            "created": "2013-09-10T13:57:02.480+0000",
            "lastUpdated": "2013-09-10T13:57:02.480+0000"
        }
    ],
    "userGroupAccesses": []
}
```


A more tailored response can be obtained by specifying specific fields in the request. An example is provided below, which would return more detailed information about each object on a users dashboard.

```
/api/dashboards/vQFhmLJU5sK/?fields=:all,dashboardItems[:all]
```

Searching dashboards

When a user is building a dashboard it is convenient to be able to search for various analytical resources using the */dashboards/q* resource. This resource lets you search for matches on the name property of the following objects: charts, maps, report tables, users, reports and resources. You can do a search by making a *GET* request on the following resource URL pattern, where my-query should be replaced by the preferred search query:

```
/api/dashboards/q/my-query.json
```

For example, this query:

```
/api/dashboards/q/ma?count=6&maxCount=20&max=CHART&max=MAP
```

Will search for the following:

- Analytical object name contains the string "ma"
- Return up to 6 of each type
- For CHART and MAP types, return up to 20 items

dashboards/q query parameters

Query parameter	Description	Type	Default
count	The number of items of each type to return	Positive integer	6
maxCount	The number of items of max types to return	Positive integer	25
max	The type to return the maxCount for	String [CHART|MAP|REPORT_TABLE|USER|REPORT|RESOURCE|VISUALIZATION]	N/A

JSON and XML response formats are supported. The response in JSON format will contain references to matching resources and counts of how many matches were found in total and for each type of resource. It will look similar to this:

```
{
    "charts": [
        {
            "name": "ANC: 1-3 dropout rate Yearly",
            "id": "LW0O27b7TdD"
        },
        {
            "name": "ANC: 1 and 3 coverage Yearly",
            "id": "UlfTKWZWV4u"
        },
        {
            "name": "ANC: 1st and 3rd trends Monthly",
            "id": "gnROK20DfAA"
        }
    ],
    "visualizations": [
```

```
        {
            "name": "ANC: ANC 3 Visits Cumulative Numbers",
            "id": "arf9OiyV7df",
            "type": "LINE"
        },
        {
            "name": "ANC: 1st and 2rd trends Monthly",
            "id": "jkf6OiyV7el",
            "type": "PIVOT_TABLE"
        }
    ],
    "maps": [
        {
            "name": "ANC: 1st visit at facility (fixed) 2013",
            "id": "YOEGBvxjAY0"
        },
        {
            "name": "ANC: 3rd visit coverage 2014 by district",
            "id": "ytkZY3ChM6J"
        }
    ],
    "reportTables": [
        {
            "name": "ANC: ANC 1 Visits Cumulative Numbers",
            "id": "tWg9OiyV7mu"
        }
    ],
    "reports": [
        {
            "name": "ANC: 1st Visit Cumulative Chart",
            "id": "Kvg1AhYHM8Q"
        },
        {
            "name": "ANC: Coverages This Year",
            "id": "qYVNH1wkZR0"
        }
    ],
    "searchCount": 8,
    "chartCount": 3,
    "mapCount": 2,
    "reportTableCount": 1,
    "reportCount": 2,
    "userCount": 0,
    "patientTabularReportCount": 0,
    "resourceCount": 0
}
```


Creating, updating and removing dashboards

Creating, updating and deleting dashboards follow standard REST semantics. In order to create a new dashboard you can make a *POST* request to the `/api/dashboards` resource. From a consumer perspective it might be convenient to first create a dashboard and later add items to it. JSON and XML formats are supported for the request payload. To create a dashboard with the name "My dashboard" you can use a payload in JSON like this:

```
{
  "name": "My dashboard"
}
```

To update, e.g. rename, a dashboard, you can make a *PUT* request with a similar request payload the same api/dashboards resource.

To remove a dashboard, you can make a *DELETE* request to the specific dashboard resource similar to this:

```
/api/dashboards/vQFhmLJU5sK
```

Adding, moving and removing dashboard items and content

In order to add dashboard items a consumer can use the `/api/dashboards/<dashboard-id>/items/content` resource, where \<dashboard-id> should be replaced by the relevant dashboard identifier. The request must use the *POST* method. The URL syntax and parameters are described in detail in the following table.

Items content parameters

Query parameter	Description	Options
type	Type of the resource to be represented by the dashboard item	chart | visualization | map | reportTable | users | reports | reportTables | resources | patientTabularReports | app
id	Identifier of the resource to be represented by the dashboard item	Resource identifier

A *POST* request URL for adding a chart to a specific dashboard could look like this, where the last id query parameter value is the chart resource identifier:

```
/api/dashboards/vQFhmLJU5sK/items/content?type=chart&id=LW0O27b7TdD
```

When adding resource of type map, chart, report table and app, the API will create and add a new item to the dashboard. When adding a resource of type users, reports, report tables and resources, the API will try to add the resource to an existing dashboard item of the same type. If no item of same type or no item of same type with less than eight resources associated with it exists, the API will create a new dashboard item and add the resource to it.

In order to move a dashboard item to a new position within the list of items in a dashboard, a consumer can make a *POST* request to the following resource URL, where `<dashboard-id>` should be replaced by the identifier of the dashboard, `<item-id>` should be replaced by the identifier of the dashboard item and `<index>` should be replaced by the new position of the item in the dashboard, where the index is zero-based:

```
/api/dashboards/<dashboard-id>/items/<item-id>/position/<index>
```

To remove a dashboard item completely from a specific dashboard a consumer can make a *DELETE* request to the below resource URL, where `<dashboard-id>` should be replaced by the identifier of the dashboard and `<item-id>` should be replaced by the identifier of the dashboard item. The dashboard item identifiers can be retrieved through a GET request to the dashboard resource URL.

```
/api/dashboards/<dashboard-id>/items/<item-id>
```

To remove a specific content resource within a dashboard item a consumer can make a *DELETE* request to the below resource URL, where `<content-resource-id>` should be replaced by the identifier of a resource associated with the dashboard item; e.g. the identifier of a report or a user. For instance, this can be used to remove a single report from a dashboard item of type reports, as opposed to removing the dashboard item completely:

```
/api/dashboards/<dashboard-id>/items/<item-id>/content/<content-resource-id>
```

Visualization

The Visualization API is designed to help clients to interact with charts and pivot/report tables. The endpoints of this API are used by the Data Visualization application which allows the creation, configuration and management of charts and pivot tables based on the client's definitions. The main idea is to enable clients and users to have a unique and centralized API providing all types of charts and pivot tables as well as specific parameters and configuration for each type of visualization.

This API was introduced with the expectation to unify both `charts` and `reportTables` APIs and entirely replace them in favour of the `visualizations` API (which means that the usage of `charts` and `reportTables` APIs should be avoided). In summary, the following resources/APIs:

```
/api/charts, /api/reportTables
```

are being replaced by

```
/api/visualizations
```

> **Note**
>
> New applications and clients should avoid using the `charts` and `reportTables` APIs because they are deprecated. Use the `visualizations` API instead.

A Visualization object is composed of many attributes (some of them related to charts and others related to pivot tables), but the most important ones responsible to reflect the core information of the object are: *"id", "name", "type", "dataDimensionItems", "columns", "rows" and "filters".*

The root endpoint of the API is `/api/visualizations`, and the list of current attributes and elements are described in the table below.

Visualization attributes

Field	Description
id	The unique identifier.
code	A custom code to identify the Visualization.
name	The name of the Visualization

Field	Description
type	The type of the Visualization. The valid types are: COLUMN, STACKED_COLUMN, BAR, STACKED_BAR, LINE, AREA, PIE, RADAR, GAUGE, YEAR_OVER_YEAR_LINE YEAR_OVER_YEAR_COLUMN, SINGLE_VALUE, PIVOT_TABLE.
title	A custom title.
subtitle	A custom subtitle.
description	Defines a custom description for the Visualization.
created	The date/time of the Visualization creation.
startDate	The beginning date used during the filtering.
endDate	The ending date used during the filtering.
sortOrder	The sorting order of this Visualization. Integer value.
user	An object representing the creator of the Visualization.
publicAccess	Sets the permissions for public access.
displayDensity	The display density of the text.
fontSize	The font size of the text.
fontStyle	Custom font styles for: visualizationTitle, visualizationSubtitle, horizontalAxisTitle, verticalAxisTitle, targetLineLabel, baseLineLabel, seriesAxisLabel, categoryAxisLabel, legend.
relativePeriods	An object representing the relative periods used in the analytics query.
legendSet	An object representing the definitions for the legend.
legendDisplayStyle	The legend's display style. It can be: FILL or TEXT.
legendDisplayStrategy	The legend's display style. It can be: FIXED or BY_DATA_ITEM.
aggregationType	Determines how the values in the pivot table are aggregated. Valid options: SUM, AVERAGE, AVERAGE_SUM_ORG_UNIT, LAST, LAST_AVERAGE_ORG_UNIT, FIRST, FIRST_AVERAGE_ORG_UNIT, COUNT, STDDEV, VARIANCE, MIN, MAX, NONE, CUSTOM or DEFAULT.
regressionType	A valid regression type: NONE, LINEAR, POLYNOMIAL or LOESS.
targetLineValue	The chart target line. Accepts a Double type.
targetLineLabel	The chart target line label.
rangeAxisLabel	The chart vertical axis (y) label/title.
domainAxisLabel	The chart horizontal axis (x) label/title.
rangeAxisMaxValue	The chart axis maximum value. Values outside of the range will not be displayed.
rangeAxisMinValue	The chart axis minimum value. Values outside of the range will not be displayed.
rangeAxisSteps	The number of axis steps between the minimum and maximum values.
rangeAxisDecimals	The number of decimals for the axes values.
baseLineValue	A chart baseline value.
baseLineLabel	A chart baseline label.
digitGroupSeparator	The digit group separator. Valid values: COMMA, SPACE or NONE.
topLimit	The top limit set for the Pivot table.
measureCriteria	Describes the criteria applied to this measure.
percentStackedValues	Uses stacked values or not. More likely to be applied for graphics/charts. Boolean value.

Field	Description
noSpaceBetweenColumns	Show/hide space between columns. Boolean value.
regression	Indicates whether the Visualization contains regression columns. More likely to be applicable to Pivot/Report. Boolean value.
externalAccess	Indicates whether the Visualization is available as external read-only. Boolean value.
userOrganisationUnit	Indicates if the user has an organisation unit. Boolean value.
userOrganisationUnitChildren	Indicates if the user has a children organisation unit. Boolean value.
userOrganisationUnitGrandChildren	Indicates if the user has a grand children organisation unit . Boolean value.
reportingParams	Object used to define boolean attributes related to reporting.
rowTotals	Displays (or not) the row totals. Boolean value.
colTotals	Displays (or not) the columns totals. Boolean value.
rowSubTotals	Displays (or not) the row sub-totals. Boolean value.
colSubTotals	Displays (or not) the columns sub-totals. Boolean value.
cumulativeValues	Indicates whether the visualization is using cumulative values. Boolean value.
hideEmptyColumns	Indicates whether to hide columns with no data values. Boolean value.
hideEmptyRows	Indicates whether to hide rows with no data values. Boolean value.
completedOnly	Indicates whether to hide columns with no data values. Boolean value.
skipRounding	Apply or not rounding. Boolean value.
showDimensionLabels	Shows the dimension labels or not. Boolean value.
hideTitle	Hides the title or not. Boolean value.
hideSubtitle	Hides the subtitle or not. Boolean value.
hideLegend	Show/hide the legend. Very likely to be used by charts. Boolean value.
showHierarchy	Displays (or not) the organisation unit hierarchy names. Boolean value.
showData	Used by charts to hide or not data/values within the rendered model. Boolean value.
lastUpdatedBy	Object that represents the user that applied the last changes to the Visualization.
lastUpdated	The date/time of the last time the Visualization was changed.
favorites	List of user ids who have marked this object as a favorite.
subscribers	List of user ids who have subscribed to this Visualization.
translations	Set of available object translation, normally filtered by locale.

Retrieving visualizations

To retrieve a list of all existing visualizations, in JSON format, with some basic information (including identifier, name and pagination) you can make a GET request to the URL below. You should see a list of all public/shared visualizations plus your private ones.

```
GET /api/visualizations.json
```

If you want to retrieve the JSON definition of a specific Visualization you can add its respective identifier to the URL:

```
GET /api/visualizations/hQxZGXqnLS9.json
```

The following representation is an example of a response in JSON format (for brevity, certain information has been removed). For the complete schema, please use `GET /api/schemas/visualization`.

```
{
    "lastUpdated": "2020-02-06T11:57:09.678",
    "href": "http://my-domain/dhis/api/visualizations/hQxZGXqnLS9",
    "id": "hQxZGXqnLS9",
    "created": "2017-05-19T17:22:00.785",
    "name": "ANC: ANC 1st visits last 12 months cumulative values",
    "publicAccess": "rw------",
    "userOrganisationUnitChildren": false,
    "type": "LINE",
    "access": {},
    "reportingParams": {
        "parentOrganisationUnit": false,
        "reportingPeriod": false,
        "organisationUnit": false,
        "grandParentOrganisationUnit": false
    },
    "dataElementGroupSetDimensions": [],
    "attributeDimensions": [],
    "yearlySeries": [],
    "filterDimensions": ["dx"],
    "columns": [
        {
            "id": "ou"
        }
    ],
    "dataElementDimensions": [],
    "categoryDimensions": [],
    "rowDimensions": ["pe"],
    "columnDimensions": ["ou"],
    "dataDimensionItems": [
        {
            "dataDimensionItemType": "DATA_ELEMENT",
            "dataElement": {
                "id": "fbfJHSPpUQD"
            }
        }
    ],
    "filters": [
        {
            "id": "dx"
        }
    ],
    "rows": [
        {
            "id": "pe"
        }
    ]
}
```

A more tailored response can be obtained by specifying, in the URL, the fields you want to extract. Ie.:

```
GET /api/visualizations/hQxZGXqnLS9.json?fields=interpretations
```

will return

```
{
    "interpretations": [
        {
            "id": "Lfr8I2RPU0C"
        },
        {
            "id": "JuwgdJlJPGb"
        },
        {
            "id": "WAoU2rSpyZp"
        }
    ]
}
```

As seen, the GET above will return only the interpretations related to the given identifier (in this case hQxZGXqnLS9).

Creating, updating and removing visualizations

These operations follow the standard *REST* semantics. A new Visualization can be created through a POST request to the `/api/visualizations` resource with a valid JSON payload. An example of payload could be:

```
{
    "columns": [
        {
            "dimension": "J5jldMd8OHv",
            "items": [
                {
                    "name": "CHP",
                    "id": "uYxK4wmcPqA",
                    "displayName": "CHP",
                    "displayShortName": "CHP",
                    "dimensionItemType": "ORGANISATION_UNIT_GROUP"
                },
                {
                    "name": "Hospital",
                    "id": "tDZVQ1WtwpA",
                    "displayName": "Hospital",
                    "displayShortName": "Hospital",
                    "dimensionItemType": "ORGANISATION_UNIT_GROUP"
                }
            ]
        }
    ],
    "rows": [
        {
            "dimension": "SooXFOUnciJ",
            "items": [
                {
                    "name": "DOD",
                    "id": "B0bjKC0szQX",
                    "displayName": "DOD",
                    "displayShortName": "DOD",
```

```
                    "dimensionItemType": "CATEGORY_OPTION_GROUP"
                },
                {
                    "name": "CDC",
                    "id": "OK2Nr4wdfrZ",
                    "displayName": "CDC",
                    "displayShortName": "CDC",
                    "dimensionItemType": "CATEGORY_OPTION_GROUP"
                }
            ]
        }
    ],
    "filters": [
        {
            "dimension": "ou",
            "items": [
                {
                    "name": "Sierra Leone",
                    "id": "ImspTQPwCqd",
                    "displayName": "Sierra Leone",
                    "displayShortName": "Sierra Leone",
                    "dimensionItemType": "ORGANISATION_UNIT"
                },
                {
                    "name": "LEVEL-1",
                    "id": "LEVEL-H1KlN4QIauv",
                    "displayName": "LEVEL-1"
                }
            ]
        }
    ],
    "name": "HIV Cases Monthly",
    "description": "Cases of HIV across the months",
    "category": "XY1vwCQskjX",
    "showDimensionLabels": true,
    "hideEmptyRows": true,
    "hideEmptyColumns": true,
    "skipRounding": true,
    "aggregationType": "SUM",
    "regressionType": "LINEAR",
    "type": "PIVOT_TABLE",
    "numberType": "VALUE",
    "measureCriteria": "Some criteria",
    "showHierarchy": true,
    "completedOnly": true,
    "displayDensity": "NORMAL",
    "fontSize": "NORMAL",
    "digitGroupSeparator": "SPACE",
    "legendDisplayStyle": "FILL",
    "legendDisplayStrategy": "FIXED",
    "hideEmptyRowItems": "BEFORE_FIRST_AFTER_LAST",
    "regression": false,
    "cumulative": true,
    "sortOrder": 1,
    "topLimit": 2,
    "rowTotals": true,
    "colTotals": true,
    "hideTitle": true,
    "hideSubtitle": true,
    "hideLegend": true,
    "showData": true,
    "baseLineLabel": "A base label",
    "targetLineLabel": "A target label",
```

```
    "targetLineValue": 45.5,
    "baseLineValue": 19.99,
    "percentStackedValues": true,
    "noSpaceBetweenColumns": true,
    "rowSubTotals": true,
    "colSubTotals": true,
    "domainAxisLabel": "A domain axis label",
    "rangeAxisLabel": "A range axis label",
    "rangeAxisMaxValue": 123.65,
    "rangeAxisMinValue": 33.89,
    "rangeAxisSteps": 5,
    "rangeAxisDecimals": 10,
    "userOrgUnitType": "TEI_SEARCH",
    "externalAccess": false,
    "publicAccess": "--------",
    "reportingParams": {
        "reportingPeriod": true,
        "organisationUnit": true,
        "parentOrganisationUnit": true,
        "grandParentOrganisationUnit": true
    },
    "parentGraphMap": {
        "ImspTQPwCqd": ""
    },
    "access": {
        "read": true,
        "update": true,
        "externalize": true,
        "delete": false,
        "write": true,
        "manage": false
    },
    "optionalAxes": [
        {
            "dimensionalItem": "fbfJHSPpUQD",
            "axis": 1
        },
        {
            "dimensionalItem": "cYeuwXTCPkU",
            "axis": 2
        }
    ],
    "relativePeriods": {
        "thisYear": false,
        "quartersLastYear": true,
        "last52Weeks": false,
        "thisWeek": false,
        "lastMonth": false,
        "last14Days": false,
        "biMonthsThisYear": false,
        "monthsThisYear": false,
        "last2SixMonths": false,
        "yesterday": false,
        "thisQuarter": false,
        "last12Months": false,
        "last5FinancialYears": false,
        "thisSixMonth": false,
        "lastQuarter": false,
        "thisFinancialYear": false,
        "last4Weeks": false,
        "last3Months": false,
        "thisDay": false,
        "thisMonth": false,
```

```
        "last5Years": false,
        "last6BiMonths": false,
        "last4BiWeeks": false,
        "lastFinancialYear": false,
        "lastBiWeek": false,
        "weeksThisYear": false,
        "last6Months": false,
        "last3Days": false,
        "quartersThisYear": false,
        "monthsLastYear": false,
        "lastWeek": false,
        "last7Days": false,
        "thisBimonth": false,
        "lastBimonth": false,
        "lastSixMonth": false,
        "thisBiWeek": false,
        "lastYear": false,
        "last12Weeks": false,
        "last4Quarters": false
    },
    "user": {},
    "yearlySeries": ["THIS_YEAR"],
    "userGroupAccesses": [
        {
            "access": "rwx-----",
            "userGroupUid": "ZoHNWQajIoe",
            "displayName": "Bo District M&E officers",
            "id": "ZoHNWQajIoe"
        }
    ],
    "userAccesses": [
        {
            "access": "--------",
            "displayName": "John Barnes",
            "id": "DXyJmlo9rge",
            "userUid": "DXyJmlo9rge"
        }
    ],
    "legendSet": {
        "name": "Death rate up",
        "id": "ham2eIDJ9k6",
        "legends": [
            {
                "startValue": 1,
                "endValue": 2,
                "color": "red",
                "image": "some-image"
            },
            {
                "startValue": 2,
                "endValue": 3,
                "color": "blue",
                "image": "other-image"
            }
        ]
    },
    "fontStyle": {
        "visualizationTitle": {
            "font": "VERDANA",
            "fontSize": 16,
            "bold": true,
            "italic": false,
            "underline": false,
```

```
            "textColor": "#3a3a3a",
            "textAlign": "LEFT"
        },
        "horizontalAxisTitle": {
            "font": "ROBOTO",
            "fontSize": 12,
            "bold": false,
            "italic": true,
            "underline": false,
            "textColor": "#2a2a2a",
            "textAlign": "CENTER"
        },
        "categoryAxisLabel": {
            "font": "ROBOTO",
            "fontSize": 12,
            "bold": false,
            "italic": true,
            "underline": false,
            "textColor": "#dedede",
            "textAlign": "CENTER"
        },
        "targetLineLabel": {
            "font": "ARIAL",
            "fontSize": 12,
            "bold": false,
            "italic": true,
            "underline": false,
            "textColor": "#dedede",
            "textAlign": "CENTER"
        }
    }
}
```

To update a specific Visualization, you can send a PUT request to the same `/api/visualizations` resource with a similar payload PLUS the respective Visualization's identifier, ie.:

```
PUT /api/visualizations/hQxZGXqnLS9
```

Finally, to delete an existing Visualization, you can make a DELETE request specifying the identifier of the Visualization to be removed, as shown:

```
DELETE /api/visualizations/hQxZGXqnLS9
```

Data items

This endpoint allows the user to query data related to a few different dimensional items. These items are: `INDICATOR`, `DATA_ELEMENT`, `DATA_SET`, `PROGRAM_INDICATOR`, `PROGRAM_DATA_ELEMENT`, `PROGRAM_ATTRIBUTE`. The endpoint supports only GET requests and, as other endpoints, can return responses in JSON or XML format.

The URL is `/api/dataItems` and as you can imagine, it is able to retrieve different objects through the same endpoint in the same GET request. For this reason, some queriable attributes available will differ depending on the dimensional item(s) being queried.

To understand the statement above let's have a look at the followings request examples:

1. `GET /api/dataItems?filter=dimensionItemType:eq:DATA_ELEMENT&filter=valueType:eq:TEXT` In this example the item type `DATA_ELEMENT` has a `valueType` attribute which can be used in the query.
2. `GET /api/dataItems?pageSize=50&order=displayName:asc&filter=dimensionItemType:eq:PROGRAM_INDICATOR&fil`

Here, the `PROGRAM_INDICATOR` allows filtering by `programId`.

So, based on the examples `1)` and `2)` if you try filtering a `DATA_ELEMENT` by `programId` or filter a `PROGRAM_INDICATOR` by `valueType`, you should get no results. In other words, the filter will be applied only when the attribute actually exists for the respective data item.

Another important aspect to be highlighted is that this endpoint does NOT follows the same querying standards as other existing endpoints, like Metadata object filter for example. As a consequence, it supports a smaller set of features and querying. The main reason for that is the need for querying multiple different items that have different relationships, which is not possible using the existing filtering components (used by the others endpoints).

Possible endpoint responses

Base on the `GET` request/query, a few different responses are possible. Below we are summarizing each possibility.

Results found (HTTP status code 200)


```
{
  "pager": {
    "page": 1,
    "pageCount": 27,
    "total": 1339,
    "pageSize": 50,
    "nextPage": "https://play.dhis2.org/dev/api/36/dataItems?
page=2&filter=displayName:ilike:a&filter=id:eq:nomatch&rootJunction=OR&displayName:asc=&paging=true"
  },
  "dataItems": [
    {
      "simplifiedValueType": "TEXT",
      "displayName": "TB program Gender",
      "displayShortName": "TB prog. Gen.",
      "valueType": "TEXT",
      "name": "TB program Gender",
      "shortName": ""TB prog. Gen.",
      "id": "ur1Edk5Oe2n.cejWyOfXge6",
      "programId": "ur1Edk5Oe2n",
      "dimensionItemType": "PROGRAM_ATTRIBUTE"
    },
    ...
  ]
}
```


Results not found (HTTP status code 200)


```
{
  "pager": {
```

```
    "page": 1,
    "pageCount": 1,
    "total": 0,
    "pageSize": 50
  },
  "dataItems": []
}
```

Invalid query (HTTP status code 409)

```
{
  "httpStatus": "Conflict",
  "httpStatusCode": 409,
  "status": "ERROR",
  "message": "Unable to parse element `INVALID_TYPE` on filter `dimensionItemType`. The values
available are: [INDICATOR, DATA_ELEMENT, DATA_ELEMENT_OPERAND, DATA_SET, PROGRAM_INDICATOR,
PROGRAM_DATA_ELEMENT, PROGRAM_ATTRIBUTE]",
  "errorCode": "E2016"
}
```

Unhandled error (HTTP status code 500)

```
{
  "httpStatus": "Internal Server Error",
  "httpStatusCode": 500,
  "status": "ERROR"
}
```

Pagination

This endpoint also supports pagination as a default option. If needed, you can disable pagination by adding `paging=false` to the GET request. ie.: `/api/dataItems?filter=dimensionItemType:in:[INDICATOR]&paging=false`.

Here is an example of a payload when the pagination is enabled. Remember that pagination is the default option and does not need to be explicitly set.

```
{
  "pager": {
    "page": 1,
    "pageCount": 20,
    "total": 969,
    "pageSize": 50,
    "nextPage": "https://play.dhis2.org/dev/api/dataItems?page=2&filter=dimensionItemType:in:
[INDICATOR]"
  },
  "dataItems": [...]
}
```

> **Note**
>
> For elements where there is an associated Program, the program name should also be returned as part of the element name (as a prefix). The only exception is `Program Indicators`. We will not prefix the element

name in this case, in order to keep the same behavior of existing endpoints.

The /dataItems endpoint will bring only data items that are defined as aggregatable type. The current list of valid aggregatable types is: `TEXT`, `LONG_TEXT`, `LETTER`, `BOOLEAN`, `TRUE_ONLY`, `NUMBER`, `UNIT_INTERVAL`, `PERCENTAGE`, `INTEGER`, `INTEGER_POSITIVE`, `INTEGER_NEGATIVE`, `INTEGER_ZERO_OR_POSITIVE`, `COORDINATE`.

Even though the response returns a few different attributes, the filtering can only be applied to specific ones: `displayName`, `name`, `valueType`, `id`, `dimensionItemType`, `programId`.

The `order` will be considered invalid if it's set on top of `name` (ie.: order=*name:asc*) and a `filter` is set to `displayName` (ie.: filter=*displayName:ilike:aName*), and vice-versa.

Response attributes

Now that we have a good idea of the main features and usage of this endpoint let's have a look in the list of attributes returned in the response.

Data items attributes

Field	Description
id	The unique identifier.
code	A custom code to identify the dimensional item.
name	The name given for the item.
displayName	The display name defined.
shortName	The short name given for the item.
displayShortName	The display short name defined.
dimensionItemType	The dimension type. Possible types: INDICATOR, DATA_ELEMENT, REPORTING_RATE, PROGRAM_INDICATOR, PROGRAM_DATA_ELEMENT, PROGRAM_ATTRIBUTE.
valueType	The item value type (more specific definition). Possible types: TEXT, LONG_TEXT, LETTER, BOOLEAN, TRUE_ONLY, UNIT_INTERVAL, PERCENTAGE, INTEGER, INTEGER_POSITIVE, INTEGER_NEGATIVE, INTEGER_ZERO_OR_POSITIVE, COORDINATE
simplifiedValueType	The genereal representation of a value type. Valid values: NUMBER, BOOLEAN, DATE, FILE_RESOURCE, COORDINATE, TEXT
programId	The associated programId.

Analytics

To access analytical, aggregated data in DHIS2 you can work with the *analytics* resource. The analytics resource is powerful as it lets you query and retrieve data aggregated along all available data dimensions. For instance, you can ask the analytics resource to provide the aggregated data values for a set of data elements, periods and organisation units. Also, you can retrieve the aggregated data for a combination of any number of dimensions based on data elements and organisation unit group sets.

```
/api/33/analytics
```

Request query parameters

The analytics resource lets you specify a range of query parameters:

Query parameters

Query parameter	Required	Description	Options (default first)
dimension	Yes	Dimensions and dimension items to be retrieved, repeated for each.	Any dimension
filter	No	Filters and filter items to apply to the query, repeated for each.	Any dimension
aggregationType	No	Aggregation type to use in the aggregation process.	SUM | AVERAGE | AVERAGE_SUM_ORG_UNIT | LAST | LAST_AVERAGE_ORG_UNIT | COUNT | STDDEV | VARIANCE | MIN | MAX
measureCriteria	No	Filters for the data/measures.	EQ | GT | GE | LT | LE
preAggregationMeasureCriteria	No	Filters for the data/measure, applied before aggregation is performed.	EQ | GT | GE | LT | LE
startDate	No	Start date for a date range. Will be applied as a filter. Can not be used together with a period dimension or filter.	Date
endDate	No	End date for date range. Will be applied as a filter. Can not be used together with a period dimension or filter.	Date
skipMeta	No	Exclude the metadata part of the response (improves performance).	false | true
skipData	No	Exclude the data part of the response.	false | true
skipRounding	No	Skip rounding of data values, i.e. provide full precision.	false | true
hierarchyMeta	No	Include names of organisation unit ancestors and hierarchy paths of organisation units in the metadata.	false | true
ignoreLimit	No	Ignore limit on max 50 000 records in response - use with care.	false | true
tableLayout	No	Use plain data source or table layout for the response.	false | true
hideEmptyRows	No	Hides empty rows in response, applicable when table layout is true.	false | true
hideEmptyColumns	No	Hides empty columns in response, applicable when table layout is true.	false | true
showHierarchy	No	Display full org unit hierarchy path together with org unit name.	false | true
includeNumDen	No	Include the numerator and denominator used to calculate the value in the response.	false | true

Query parameter	Required	Description	Options (default first)
includeMetadataDetails	No	Include metadata details to raw data response.	false | true
displayProperty	No	Property to display for metadata.	NAME | SHORTNAME
outputIdScheme	No	Identifier scheme to use for metadata items the query response, can be identifier, code or attributes.	UID | CODE |NAME| ATTRIBUTE:<ID>
inputIdScheme	No	Identifier scheme to use for metadata items in the query request, can be an identifier, code or attributes.	UID | CODE | ATTRIBUTE:<ID>
approvalLevel	No	Include data which has been approved at least up to the given approval level, refers to identifier of approval level.	Identifier of approval level
relativePeriodDate	No	Date used as basis for relative periods.	Date.
userOrgUnit	No	Explicitly define the user org units to utilize, overrides organisation units associated with the current user, multiple identifiers can be separated by semicolon.	Organisation unit identifiers.
columns	No	Dimensions to use as columns for table layout.	Any dimension (must be query dimension)
rows	No	Dimensions to use as rows for table layout.	Any dimension (must be query dimension)
order	No	Specify the ordering of rows based on value.	ASC | DESC
timeField	No	The time field to base event aggregation on. Applies to event data items only. Can be a predefined option or the ID of an attribute or data element with a time-based value type.	EVENT_DATE | ENROLLMENT_DATE | INCIDENT_DATE | DUE_DATE | COMPLETED_DATE | CREATED | LAST_UPDATED | <Attribute ID> | <Data element ID>
orgUnitField	No	The organisation unit field to base event aggregation on. Applies to event data items only. Can be the ID of an attribute or data element with the Organisation unit value type. The default option is specified as omitting the query parameter.	<Attribute ID> | <Data element ID>

The *dimension* query parameter defines which dimensions should be included in the analytics query. Any number of dimensions can be specified. The dimension parameter should be repeated for each dimension to include in the query response. The query response can potentially contain aggregated values for all combinations of the specified dimension items.

The *filter* parameter defines which dimensions should be used as filters for the data retrieved in the analytics query. Any number of filters can be specified. The filter parameter should be repeated for each filter to use in the query. A filter differs from a dimension in that the filter

dimensions will not be part of the query response content, and that the aggregated values in the response will be collapsed on the filter dimensions. In other words, the data in the response will be aggregated on the filter dimensions, but the filters will not be included as dimensions in the actual response. As an example, to query for certain data elements filtered by the periods and organisation units you can use the following URL:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&filter=pe:2014Q1;2014Q2
  &filter=ou:O6uvpzGd5pu;lc3eMKXaEfw
```

The *aggregationType* query parameter lets you define which aggregation operator should be used for the query. By default, the aggregation operator defined for data elements included in the query will be used. If your query does not contain any data elements but does include data element groups, the aggregation operator of the first data element in the first group will be used. The order of groups and data elements is undefined. This query parameter allows you to override the default and specify a specific aggregation operator. As an example, you can set the aggregation operator to "count" with the following URL:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2014Q1&dimension=ou:O6uvpzGd5pu
  &aggregationType=COUNT
```

The *measureCriteria* query parameter lets you filter out ranges of data records to return. You can instruct the system to return only records where the aggregated data value is equal, greater than, greater or equal, less than or less or equal to certain values. You can specify any number of criteria on the following format, where *criteria* and *value* should be substituted with real values:

```
/api/33/analytics?measureCriteria=criteria:value;criteria:value
```

As an example, the following query will return only records where the data value is greater or equal to 6500 and less than 33000:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&dimension=pe:2014
  &dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw&measureCriteria=GE:6500;LT:33000
```

Similar to *measureCriteria*, the *preAggregationMeasureCriteria* query parameter lets you filter out data, only before aggregation is performed. For example, the following query only aggregates data where the original value is within the criteria defined:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&dimension=pe:2014
  &dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw&preAggregationMeasureCriteria=GE:10;LT:100
```

The *startDate* and *endDate* parameters can be used to specify a custom date range to aggregate over. When specifying a date range you can not specify relative nor fixed periods as dimension or filter. The date range will filter the analytics response. You can use it like this:

```
/api/33/analytics.json?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &dimension=ou:ImspTQPwCqd&startDate=2018-01-01&endDate=2018-06-01
```

In order to have the analytics resource generate the data in the shape of a ready-made table, you can provide the *tableLayout* parameter with true as value. Instead of generating a plain, normalized data source, the analytics resource will now generate the data in a table layout. You can use the *columns* and *rows* parameters with dimension identifiers separated by semi-colons as values to indicate which ones to use as table columns and rows. The column and rows dimensions must be present as a data dimension in the query (not a filter). Such a request can look like this:

```
/api/33/analytics.html?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&dimension=pe:2014Q1;2014Q2
  &dimension=ou:O6uvpzGd5pu&tableLayout=true&columns=dx;ou&rows=pe
```

The *order* parameter can be used for analytics resource to generate ordered data. The data will be ordered in ascending (or descending) order of values. An example request for ordering the values in descending order is:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:LAST_12_MONTHS
  &dimension=ou:O6uvpzGd5pu&order=DESC
```

Dimensions and items

DHIS2 features a multi-dimensional data model with several fixed and dynamic data dimensions. The fixed dimensions are the data element, period (time) and organisation unit dimension. You can dynamically add dimensions through categories, data element group sets and organisation unit group sets. The table below displays the available data dimensions in DHIS2. Each data dimension has a corresponding *dimension identifier*, and each dimension can have a set of *dimension items*:

Dimensions and dimension items

Dimension	Dimension id	Dimension items
Data elements, indicators, data set reporting rate metrics, data element operands, program indicators, program data elements, program attributes, validation rules	dx	Data element, indicator, data set reporting rate metrics, data element operand, program indicator, program attribute identifiers, keyword DE_GROUP-<group-id>, IN_GROUP-<group-id>, use <dataelement-id>.<optioncombo-id> for data element operands, <program-id>.<dataelement-id> for program data elements, <program-id>.<attribute-id> for program attributes, <validationrule-id> for validation results.
Periods (time)	pe	ISO periods and relative periods, see "date and period format"
Organisation unit hierarchy	ou	Organisation unit identifiers, and keywords USER_ORGUNIT, USER_ORGUNIT_CHILDREN, USER_ORGUNIT_GRANDCHILDREN, LEVEL-<level> and OU_GROUP-<group-id>
Category option combinations	co	Category option combo identifiers (omit to get all items)
Attribute option combinations	ao	Category option combo identifiers (omit to get all items)

Dimension	Dimension id	Dimension items
Categories	<category id>	Category option identifiers (omit to get all items)
Data element group sets	<group set id>	Data element group identifiers (omit to get all items)
Organisation unit group sets	<group set id>	Organisation unit group identifiers (omit to get all items)
Category option group sets	<group set id>	Category option group identifiers (omit to get all items)

It is not necessary to be aware of which objects are used for the various dynamic dimensions when designing analytics queries. You can get a complete list of dynamic dimensions by visiting this URL in the Web API:

```
/api/33/dimensions
```

If you want to retrieve only the dimensional items for a given dynamic dimension you can use the exemple below. The pagination is disabled by default. It can be enabled by adding the pagination parameter `paging=true` to the URL.

```
/api/33/dimensions/J5jldMd8OHv/items?paging=true
```

The base URL to the analytics resource is `/api/analytics`. To request specific dimensions and dimension items you can use a query string on the following format, where `dim-id` and `dim-item` should be substituted with real values:

```
/api/33/analytics?dimension=dim-id:dim-item;dim-item&dimension=dim-id:dim-item;dim-item
```

As illustrated above, the dimension identifier is followed by a colon while the dimension items are separated by semi-colons. As an example, a query for two data elements, two periods and two organisation units can be done with the following URL:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &dimension=pe:2016Q1;2016Q2&dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw
```

To query for data broken down by category option combinations instead of data element totals you can include the category dimension in the query string, for instance like this:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &dimension=co&dimension=pe:201601&dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw
```

When selecting data elements you can also select all data elements in a group as items by using the DE_GROUP- syntax:

```
/api/33/analytics?dimension=dx:DE_GROUP-h9cuJOkOwY2
  &dimension=pe:201601&dimension=ou:O6uvpzGd5pu
```

When selecting data set reporting rates, the syntax contains a data set identifier followed by a reporting rate metric:

```
/api/33/analytics?dimension=dx:BfMAe6Itzgt.REPORTING_RATE;BfMAe6Itzgt.ACTUAL_REPORTS
  &dimension=pe:201601&dimension=ou:O6uvpzGd5pu
```

To query for program data elements (of tracker domain type) you can get those by specifying the program for each data element using the . syntax:

```
/api/33/analytics.json?dimension=dx:eBAyeGv0exc.qrur9Dvnyt5;eBAyeGv0exc.GieVkTxp4HH
  &dimension=pe:LAST_12_MONTHS&filter=ou:ImspTQPwCqd
```

To query for program attributes (tracked entity attributes) you can get those by specifying the program for each attribute using the . syntax:

```
/api/33/analytics.json?dimension=dx:IpHINAT79UW.a3kGcGDCuk6;IpHINAT79UW.UXz7xuGCEhU
  &dimension=pe:LAST_4_QUARTERS&dimension=ou:ImspTQPwCqd
```

To query for organisation unit group sets and data elements you can use the following URL. Notice how the group set identifier is used as a dimension identifier and the groups as dimension items:

```
/api/33/analytics?dimension=Bpx0589u8y0:oRVt7g429ZO;MAs88nJc9nL
  &dimension=pe:2016&dimension=ou:ImspTQPwCqd
```

To query for data elements and categories you can use this URL. Use the category identifier as a dimension identifier and the category options as dimension items:

```
/api/33/analytics?dimension=dx:s46m5MS0hxu;fClA2Erf6IO&dimension=pe:2016
  &dimension=YNZyaJHiHYq:btOyqprQ9e8;GEqzEKCHoGA&filter=ou:ImspTQPwCqd
```

To query using relative periods and organisation units associated with the current user you can use a URL like this:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &dimension=pe:LAST_12_MONTHS&dimension=ou:USER_ORGUNIT
```

When selecting organisation units for a dimension you can select an entire level optionally constrained by any number of boundary organisation units with the `LEVEL-<level>` syntax. Boundary refers to a top node in a sub-hierarchy, meaning that all organisation units at the given level below the given boundary organisation unit in the hierarchy will be included in the response, and is provided as regular organisation unit dimension items. The level value can either be a numerical level or refer to the identifier of the organisation unit level entity. A simple query for all org units at level three:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2016&dimension=ou:LEVEL-3
```

A query for level three and four with two boundary org units can be specified like this:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2016
  &dimension=ou:LEVEL-3;LEVEL-4;O6uvpzGd5pu;lc3eMKXaEf
```

When selecting organisation units you can also select all organisation units in an organisation unit group to be included as dimension items using the OU_GROUP- syntax. The organisation units in the groups can optionally be constrained by any number of boundary organisation units. Both the level and the group items can be repeated any number of times:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2016
  &dimension=ou:OU_GROUP-w0gFTTmsUcF;OU_GROUP-EYbopBOJWsW;O6uvpzGd5pu;lc3eMKXaEf
```

You can utilize identifier schemes for the metadata part of the analytics response with the outputIdScheme property like this. You can use ID, code and attributes as identifier scheme:

```
/api/33/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &dimension=pe:2017Q1;2017Q2&dimension=ou:O6uvpzGd5pu&outputIdScheme=CODE
```

A few things to be aware of when using the analytics resource are listed below.

- Data elements, indicator, data set reporting rates, program data elements and program indicators are part of a common data dimension, identified as "dx". This means that you can use any of data elements, indicators and data set identifiers together with the "dx" dimension identifier in a query.
- For the category, data element group set and organisation unit group set dimensions, all dimension items will be used in the query if no dimension items are specified.
- For the period dimension, the dimension items are ISO period identifiers and/or relative periods. Please refer to the section above called "Date and period format" for the period format and available relative periods.
- For the organisation unit dimension, you can specify the items to be the organisation unit or sub-units of the organisation unit associated with the user currently authenticated for the request using the keys `USER_ORGUNIT` or `USER_ORGUNIT_CHILDREN` as items, respectively. You can also specify organisation unit identifiers directly, or a combination of both.
- For the organisation unit dimension, you can specify the organisation hierarchy level and the boundary unit to use for the request on the format `LEVEL-<level>-<boundary-id>`; as an example `LEVEL-3-ImspTQPwCqd` implies all organisation units below the given boundary unit at level 3 in the hierarchy.
- For the organisation unit dimension, the dimension items are the organisation units and their sub-hierarchy - data will be aggregated for all organisation units below the given organisation unit in the hierarchy.
- You cannot specify dimension items for the category option combination dimension. Instead, the response will contain the items which are linked to the data values.

The dx dimension

The dx dimension is a special dimension which can contain all of the following data types.

Data dx dimension types

Type	Syntax	Description	Data source
Indicator	<indicator-id>	Indicator identifier.	Aggregated data
Indicator grop	IN_GROUP-<indicatorgroup-id>	Keyword followed by an indicator group identifier. Will include all indicators in the group in the response.	Aggregated data
Data element	<dataelement-id>	Data element identifier.	Aggregated data
Data element group	DE_GROUP-<dataelementgroup-id>	Keyword followed by a data element group identifier. Will include all data elements in the group in the response.	Aggregated data
Data element operand	<dataelement-id>.<categoryoptcombo-id>.<attributeoptcombo-id>	Data element identifier followed by one or both of category option combination and attribute option combo identifier. Wildcard "*" symbol can be used to indicate any option combination value. The attribute option combination identifier can be completely left out.	Aggregate data
Data set	<dataset-id>.<reporting-rate-metric>	Data set identifier followed by reporting rate metric. Can be REPORTING_RATE | REPORTING_RATE_ON_TIME | ACTUAL_REPORTS | ACTUAL_REPORTS_ON_TIME | EXPECTED_REPORTS.	Data set completeness registrations
Program data element	<program-id>.<dataelement-id>	Program identifier followed by data element identifier. Reads from events within the specified program.	Events from the given program
Program indicator	<programindicator-id>	Program indicator identifier. Reads from events from within the program associated with the program identifier.	Events from the program of the program indicator

Type	Syntax	Description	Data source
Validation result	<validationrule-id>	Validation rule identifier. Will include validation rule violations for the validation rule, requires that validation results are generated and persisted.	Validation results

Items from all of the various dx types can be combined in an analytics request. An example looks like this:

```
/api/33/analytics.json
  ?dimension=dx:Uvn6LCg7dVU;BfMAe6Itzgt.REPORTING_RATE;IpHINAT79UW.a3kGcGDCuk6
  &dimension=pe:LAST_12_MONTHS&filter=ou:ImspTQPwCqd
```

The group syntax can be used together with any other item as well. An example looks like this:

```
/api/33/analytics.json
  ?dimension=dx:DE_GROUP-qfxEYY9xAl6;IN_GROUP-oehv9EO3vP7;BfMAe6Itzgt.REPORTING_RATE
  &dimension=pe:LAST_12_MONTHS&filter=ou:ImspTQPwCqd
```

Data element operands can optionally specify attribute option combinations and use wildcards e.g. to specify all category option combination values:

```
/api/33/analytics.json
  ?dimension=dx:Uvn6LCg7dVU.*.j8vBiBqGf6O;Uvn6LCg7dVU.Z4oQs46iTeR
  &dimension=pe:LAST_12_MONTHS&filter=ou:ImspTQPwCqd
```

> **Tip**
>
> A great way to learn how to use the analytics API is to use the DHIS2 *pivot table* app. You can play around with pivot tables using the various dimensions and items and click Download > Plain data source > JSON to see the resulting analytics API calls in the address bar of your Web browser.

Response formats

The analytics response containing aggregate data can be returned in various representation formats. As usual, you can indicate interest in a specific format by appending a file extension to the URL, through the `Accept` HTTP header or through the `format` query parameter. The default format is JSON. The available formats and content-types are listed below.

- json (application/json)
- jsonp (application/javascript)
- xml (application/xml)
- csv (application/csv)
- html (text/html)

html+css (text/html)

- xls (application/vnd.ms-excel)

As an example, to request an analytics response in XML format you can use the following URL:

```
/api/33/analytics.xml?dimension=dx:fbfJHSPpUQD
  &dimension=pe:2016&dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw
```

The analytics responses must be retrieved using the HTTP *GET* method. This allows for direct linking to analytics responses from Web pages as well as other HTTP-enabled clients. To do functional testing we can use the cURL library. By executing this command against the demo database you will get an analytics response in JSON format:

```
curl "play.dhis2.org/demo/api/analytics.json?dimension=dx:eTDtyyaSA7f;FbKK4ofIv5R
  &dimension=pe:2016Q1;2016Q2&filter=ou:ImspTQPwCqd" -u admin:district
```

The JSON response will look like this:

```
{
    "headers": [
        {
            "name": "dx",
            "column": "Data",
            "meta": true,
            "type": "java.lang.String"
        },
        {
            "name": "pe",
            "column": "Period",
            "meta": true,
            "type": "java.lang.String"
        },
        {
            "name": "value",
            "column": "Value",
            "meta": false,
            "type": "java.lang.Double"
        }
    ],
    "height": 4,
    "metaData": {
        "pe": ["2016Q1", "2016Q2"],
        "ou": ["ImspTQPwCqd"],
        "names": {
            "2016Q1": "Jan to Mar 2016",
            "2016Q2": "Apr to Jun 2016",
            "FbKK4ofIv5R": "Measles Coverage <1 y",
            "ImspTQPwCqd": "Sierra Leone",
            "eTDtyyaSA7f": "Fully Immunized Coverage"
        }
    },
    "rows": [
        ["eTDtyyaSA7f", "2016Q2", "81.1"],
        ["eTDtyyaSA7f", "2016Q1", "74.7"],
        ["FbKK4ofIv5R", "2016Q2", "88.9"],
        ["FbKK4ofIv5R", "2016Q1", "84.0"]
    ],
```

```
    "width": 3
}
```

The response represents a table of dimensional data. The *headers* array gives an overview of which columns are included in the table and what the columns contain. The *column* property shows the column dimension identifier, or if the column contains measures, the word "Value". The *meta* property is *true* if the column contains dimension items or *false* if the column contains a measure (aggregated data values). The *name* property is similar to the column property, except it displays "value" in case the column contains a measure. The *type* property indicates the Java class type of column values.

The *height* and *width* properties indicate how many data columns and rows are contained in the response, respectively.

The *metaData periods* property contains a unique, ordered array of the periods included in the response. The *metaData ou* property contains an array of the identifiers of organisation units included in the response. The *metaData names* property contains a mapping between the identifiers used in the data response and the names of the objects they represent. It can be used by clients to substitute the identifiers within the data response with names in order to give a more meaningful view of the data table.

The *rows* array contains the dimensional data table. It contains columns with dimension items (object or period identifiers) and a column with aggregated data values. The example response above has a data/indicator column, a period column and a value column. The first column contains indicator identifiers, the second contains ISO period identifiers and the third contains aggregated data values.

Constraints and validation

There are several constraints to the input parameters you can provide to the analytics resource. If any of the constraints are violated, the API will return a *409 Conflict* response and a response message looking similar to this:

```
{
    "httpStatus": "Conflict",
    "httpStatusCode": 409,
    "status": "ERROR",
    "message": "Only a single indicator can be specified as filter",
    "errorCode": "E7108"
}
```

The `httpStatus` and `httpStatusCode` fields indicate the HTTP status and status code per the HTTP specification. The `message` field provides a human-readable description of the validation error. The `errorCode` field provides a machine-readable code which can be used by clients to handle validation errors. The possible validation errors for the aggregate analytics API are described in the table below.

Error code	Message
E7100	Query parameters cannot be null
E7101	At least one dimension must be specified
E7102	At least one data dimension item or data element group set dimension item must be specified
E7103	Dimensions cannot be specified as dimension and filter simultaneously

Error code	Message
E7104	At least one period as dimension or filter, or start and dates, must be specified
E7105	Periods and start and end dates cannot be specified simultaneously
E7106	Start date cannot be after end date
E7107	Start and end dates cannot be specified for reporting rates
E7108	Only a single indicator can be specified as filter
E7109	Only a single reporting rate can be specified as filter
E7110	Category option combos cannot be specified as filter
E7111	Dimensions cannot be specified more than once
E7112	Reporting rates can only be specified together with dimensions of type
E7113	Assigned categories cannot be specified when data elements are not specified
E7114	Assigned categories can only be specified together with data elements, not indicators or reporting rates
E7115	Data elements must be of a value and aggregation type that allow aggregation
E7116	Indicator expressions cannot contain cyclic references
E7117	A data dimension 'dx' must be specified when output format is DATA_VALUE_SET
E7118	A period dimension 'pe' must be specified when output format is DATA_VALUE_SET
E7119	An organisation unit dimension 'ou' must be specified when output format is DATA_VALUE_SET
E7120	User is not allowed to view org unit
E7121	User is not allowed to read data for object
E7122	Data approval level does not exist
E7123	Current user is constrained by a dimension but has access to no dimension items
E7124	Dimension is present in query without any valid dimension options
E7125	Dimension identifier does not reference any dimension
E7126	Column must be present as dimension in query
E7127	Row must be present as dimension in query
E7128	Query result set exceeded max limit
E7129	Program is specified but does not exist
E7130	Program stage is specified but does not exist
E7131	Query failed, likely because the query timed out

Data value set format

The analytics *dataValueSet* resource allows for returning aggregated data in the data value set format. This format represents raw data values, as opposed to data which has been aggregated along various dimensions. Exporting aggregated data as regular data values is useful for data exchange between systems when the target system contains data of finer granularity compared to what the destination system is storing.

As an example, one can specify an indicator in the target system to summarize data for multiple data elements and import this data for a single data element in the destination system. As another example, one can aggregate data collected at organisation unit level 4 in the target system to level 2 and import that data in the destination system.

You can retrieve data in the raw data value set format from the dataValueSet resource:

```
/api/33/analytics/dataValueSet
```

The following resource representations are supported:

- json (application/json)
- xml (application/xml)

When using the data value set format, exactly three dimensions must be specified as analytics dimensions with at least one dimension item each:

- Data (dx)
- Period (pe)
- Organisation unit (ou)

Any other dimension will be ignored. Filters will be applied as with regular analytics requests. Note that any data dimension type can be specified, including indicators, data elements, data element operands, data sets and program indicators.

An example request which aggregates data for specific indicators, periods and organisation units and returns it as regular data values in XML looks like this:

```
api/analytics/dataValueSet.xml?dimension=dx:Uvn6LCg7dVU;OdiHJayrsKo
  &dimension=pe:LAST_4_QUARTERS&dimension=ou:lc3eMKXaEfw;PMa2VCrupOd
```

A request which aggregates data for data element operands and uses CODE as output identifier scheme looks like the below. When defining the output identifier scheme, all metadata objects part of the response are affected:

```
api/analytics/dataValueSet.json?dimension=dx:fbfJHSPpUQD.pq2XI5kz2BY;fbfJHSPpUQD.PT59n8BQbqM
  &dimension=pe:LAST_12_MONTHS&dimension=ou:ImspTQPwCqd&outputIdScheme=CODE
```

When using attribute-based identifier schemes for export there is a risk of producing duplicate data values. The boolean query parameter duplicatesOnly can be used for debugging purposes to return only duplicates data values. This response can be used to clean up the duplicates:

```
api/analytics/dataValueSet.xml?dimension=dx:Uvn6LCg7dVU;OdiHJayrsKo
  &dimension=pe:LAST_4_QUARTERS&dimension=ou:lc3eMKXaEfw&duplicatesOnly=true
```

Raw data format

The analytics *rawData* resource allows for returning the data stored in the analytics data tables without any aggregation being performed. This is useful for clients which would like to perform aggregation and filtering on their own without having to denormalize data in the available data dimensions themselves.

```
/api/analytics/rawData
```

The following resource representations are supported:

- json (application/json)
- csv (application/csv)

This resource follows the syntax of the regular analytics resource. Only a subset of the query parameters are supported. Additionally, a *startDate* and *endDate* parameter are available. The supported parameters are listed in the table below.

Query parameters

Query parameter	Required / Notes
dimension	Yes
startDate	No / yyyy-MM-dd
endDate	No / yyyy-MM-dd
skipMeta	No
skipData	No
hierarchyMeta	No
showHierarchy	No
displayProperty	No
outputIdScheme	No
inputIdScheme	No
userOrgUnit	No

The *dimension* query parameter defines which dimensions (table columns) should be included in the response. It can optionally be constrained with items. The *filter* query parameter defines which items and dimensions (table columns) should be used as a filter for the response.

For the organisation unit dimension, the response will contain data associated with the organisation unit and all organisation units in the sub-hierarchy (children in the tree). This is different compared to the regular analytics resource, where only the explicitly selected organisation units are included.

To retrieve a response with specific data elements, specific periods, specific organisation units and all data for two custom dimensions you can issue a request like this:

```
/api/analytics/rawData.json?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU;Jtf34kNZhzP
  &dimension=J5jldMd8OHv&dimension=Bpx0589u8y0
  &dimension=pe:LAST_12_MONTHS
  &dimension=ou:O6uvpzGd5pu;fdc6uOvgoji
```

The *startDate* and *endDate* parameters allow for fetching data linked to any period between those dates. This avoids the need for defining all periods explicitly in the request:

```
/api/analytics/rawData.json?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU;Jtf34kNZhzP
  &dimension=J5jldMd8OHv&dimension=Bpx0589u8y0
  &startDate=2015-01-01&endDate=2015-12-31
  &dimension=ou:O6uvpzGd5pu;fdc6uOvgoji
```

The *filter* parameter can be used to filter a response without including that dimension as part of the response, this time in CSV format:

```
/api/analytics/rawData.csv?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU;Jtf34kNZhzP
  &filter=J5jldMd8OHv:uYxK4wmcPqA;tDZVQ1WtwpA
  &startDate=2015-01-01&endDate=2015-12-31
  &dimension=ou:O6uvpzGd5pu
```

The *outputIdScheme* parameter is useful if you want human readable data responses as it can be set to *NAME* like this:

```
/api/analytics/rawData.csv?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &filter=J5jldMd8OHv:uYxK4wmcPqA;tDZVQ1WtwpA
  &startDate=2017-01-01&endDate=2017-12-31
  &dimension=ou:O6uvpzGd5pu
  &outputIdScheme=NAME
```

The response from the *rawData* resource will look identical to the regular analytics resource; the difference is that the response contains raw, non-aggregated data, suitable for further aggregation by third-party systems.

Debugging

When debugging analytics requests it can be useful to examine the data value source of the aggregated analytics response. The *analytics/debug/sql* resource will provide an SQL statement that returns the relevant content of the datavalue table. You can produce this SQL by doing a GET request with content type "text/html" or "text/plain" like below. The dimension and filter syntax are identical to regular analytics queries:

```
/api/analytics/debug/sql?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
  &filter=pe:2016Q1;2016Q2&filter=ou:O6uvpzGd5pu
```

Event analytics

The event analytics API lets you access aggregated event data and query *events* captured in DHIS2. This resource lets you retrieve events based on a program and optionally a program stage, and lets you retrieve and filter events on any event dimensions.

```
/api/33/analytics/events
```

Dimensions and items

Event dimensions include data elements, attributes, organisation units and periods. The aggregated event analytics resource will return aggregated information such as counts or averages. The query analytics resource will simply return events matching a set of criteria and does not perform any aggregation. You can specify dimension items in the form of options from option sets and legends from legend sets for data elements and attributes which are associated with such. The event dimensions are listed in the table below.

Event dimensions

Dimension	Dimension id	Description
Data elements	<id>	Data element identifiers
Attributes	<id>	Attribute identifiers
Periods	pe	ISO periods and relative periods, see "date and period format"
Organisation units	ou	Organisation unit identifiers and keywords USER_ORGUNIT, USER_ORGUNIT_CHILDREN, USER_ORGUNIT_GRANDCHILDREN, LEVEL-<level> and OU_GROUP-<group-id>
Organisation unit group sets	<org unit group set id>	Organisation unit group set identifiers
Categories	<category id>	Category identifiers (program attribute categories only)

Request query parameters

The analytics event API lets you specify a range of query parameters.

Query parameters for both event query and aggregate analytics

Query parameter	Required	Description	Options (default first)
program	Yes	Program identifier.	Any program identifier
stage	No	Program stage identifier.	Any program stage identifier
startDate	Yes	Start date for events.	Date in yyyy-MM-dd format
endDate	Yes	End date for events.	Date in yyyy-MM-dd format

Query parameter	Required	Description	Options (default first)
dimension	Yes	Dimension identifier including data elements, attributes, program indicators, periods, organisation units and organisation unit group sets. Parameter can be repeated any number of times. Item filters can be applied to a dimension on the format <item-id>:<operator>:<filter>. Filter values are case-insensitive.	Operators can be EQ | GT | GE | LT | LE | NE | LIKE | IN
filter	No	Dimension identifier including data elements, attributes, periods, organisation units and organisation unit group sets. Parameter can be repeated any number of times. Item filters can be applied to a dimension on the format <item-id>:<operator>:<filter>. Filter values are case-insensitive.	
hierarchyMeta	No	Include names of organisation unit ancestors and hierarchy paths of organisation units in the metadata.	false | true
eventStatus	No	Specify status of events to include.	ACTIVE | COMPLETED | SCHEDULE | OVERDUE | SKIPPED
programStatus	No	Specify enrollment status of events to include.	ACTIVE | COMPLETED | CANCELLED
relativePeriodDate	string	No	Date identifier e.g: "2016-01-01". Overrides the start date of the relative period
columns	No	Dimensions to use as columns for table layout.	Any dimension (must be query dimension)
rows	No	Dimensions to use as rows for table layout.	Any dimension (must be query dimension)

Query parameters for event query analytics only

Query parameter	Required	Description	Options
ouMode	No	The mode of selecting organisation units. Default is DESCENDANTS, meaning all sub units in the hierarchy. CHILDREN refers to immediate children in the hierarchy; SELECTED refers to the selected organisation units only.	DESCENDANTS, CHILDREN, SELECTED

Query parameter	Required	Description	Options
asc	No	Dimensions to be sorted ascending, can reference event date, org unit name and code and any item identifiers.	EVENTDATE | OUNAME | OUCODE | item identifier
desc	No	Dimensions to be sorted descending, can reference event date, org unit name and code and any item identifiers.	EVENTDATE | OUNAME | OUCODE | item identifier
coordinatesOnly	No	Whether to only return events which have coordinates.	false | true
dataIdScheme	No	Id scheme to be used for data, more specifically data elements and attributes which have an option set or legend set, e.g. return the name of the option instead of the code, or the name of the legend instead of the legend ID, in the data response.	NAME | CODE | UID
page	No	The page number. Default page is 1.	Numeric positive value
pageSize	No	The page size. Default size is 50 items per page.	Numeric zero or positive value

Query parameters for aggregate event analytics only

Query parameter	Required	Description	Options
value	No	Value dimension identifier. Can be a data element or an attribute which must be of numeric value type.	Data element or attribute identifier
aggregationType	No	Aggregation type for the value dimension. Default is AVERAGE.	SUM | AVERAGE | AVERAGE_SUM_ORG_UNIT | LAST | LAST_AVERAGE_ORG_UNIT | COUNT | STDDEV | VARIANCE | MIN | MAX
showHierarchy	No	Display full org unit hierarchy path together with org unit name.	false | true
displayProperty	No	Property to display for metadata.	NAME | SHORTNAME
sortOrder	No	Sort the records on the value column in ascending or descending order.	ASC | DESC
limit	No	The maximum number of records to return. Cannot be larger than 10 000.	Numeric positive value

Query parameter	Required	Description	Options
outputType	No	Specify output type for analytical data which can be events, enrollments or tracked entity instances. The two last options apply to programs with registration only.	EVENT | ENROLLMENT | TRACKED_ENTITY_INSTANCE
collapseDataDimensions	No	Collapse all data dimensions (data elements and attributes) into a single dimension in the response.	false | true
skipMeta	No	Exclude the meta data part of the response (improves performance).	false | true
skipData	No	Exclude the data part of the response.	false | true
skipRounding	No	Skip rounding of aggregate data values.	false | true
aggregateData	No	Produce aggregate values for the data dimensions (as opposed to dimension items).	false | true
timeField	No	The time field to base event aggregation on. Applies to event data items only. Can be a predefined option or the ID of an attribute or data element having a time-based value type.	EVENT_DATE | ENROLLMENT_DATE | INCIDENT_DATE | DUE_DATE | COMPLETED_DATE | <Attribute ID> | <Data element ID>
orgUnitField	No	The organisation unit field to base event aggregation on. Applies to event data items only. Can be the ID of an attribute or data element with the Organisation unit value type. The default option is specified as omitting the query parameter.	<Attribute ID> | <Data element ID>

Query parameters for cluster event analytics only

Query parameter	Required	Description	Options
clusterSize	Yes	Size of clusters in meters.	Numeric positive value
coordinateField	No	Field to base geospatial event analytics on. Default is event. Can be set to identifiers of attributes and data elements of value type coordinate.	EVENT | <attribute-id> | <dataelement-id>
bbox	Yes	Bounding box / area of events to include in the response on the format "min longitude, min latitude, max longitude , max latitude".	String
includeClusterPoints	No	Include information about underlying points for each cluster, be careful if cluster represent a very high number of points.	false | true

Event query analytics

The *analytics/events/query* resource lets you query for captured events. This resource does not perform any aggregation, rather it lets you query and filter for information about events.

```
/api/33/analytics/events/query
```

You can specify any number of dimensions and any number of filters in a query. Dimension item identifiers can refer to any of data elements, person attributes, person identifiers, fixed and relative periods and organisation units. Dimensions can optionally have a query operator and a filter. Event queries should be on the format described below.

```
/api/33/analytics/events/query/<program-id>?startDate=yyyy-MM-dd&endDate=yyyy-MM-dd
  &dimension=ou:<ou-id>;<ou-id>&dimension=<item-id>&dimension=<item-id>:<operator>:<filter>
```

For example, to retrieve events from the "Inpatient morbidity and mortality" program between January and October 2016, where the "Gender" and "Age" data elements are included and the "Age" dimension is filtered on "18", you can use the following query:

```
/api/33/analytics/events/query/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu;fdc6uOvgoji&dimension=oZg33kd9taw&dimension=qrur9Dvnyt5:EQ:18
```

To retrieve events for the "Birth" program stage of the "Child programme" program between March and December 2016, where the "Weight" data element, filtered for values larger than 2000:

```
/api/33/analytics/events/query/IpHINAT79UW?stage=A03MvHHogjR&startDate=2016-03-01
  &endDate=2016-12-31&dimension=ou:O6uvpzGd5pu&dimension=UXz7xuGCEhU:GT:2000
```

Sorting can be applied to the query for the event date of the event and any dimensions. To sort descending on the event date and ascending on the "Age" data element dimension you can use:

```
/api/33/analytics/events/query/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu&dimension=qrur9Dvnyt5&desc=EVENTDATE&asc=qrur9Dvnyt5
```

Paging can be applied to the query by specifying the page number and the page size parameters. If page number is specified but page size is not, a page size of 50 will be used. If page size is specified but page number is not, a page number of 1 will be used. To get the third page of the response with a page size of 20 you can use a query like this:

```
/api/33/analytics/events/query/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu&dimension=qrur9Dvnyt5&page=3&pageSize=20
```

Filtering

Filters can be applied to data elements, person attributes and person identifiers. The filtering is done through the query parameter value on the following format:

```
&dimension=<item-id>:<operator>:<filter-value>
```

As an example, you can filter the "Weight" data element for values greater than 2000 and lower than 4000 like this:

```
&dimension=UXz7xuGCEhU:GT:2000&dimension=UXz7xuGCEhU:LT:4000
```

You can filter the "Age" data element for multiple, specific ages using the IN operator like this:

```
&dimension=qrur9Dvnyt5:IN:18;19;20
```

You can specify multiple filters for a given item by repeating the operator and filter components, all separated with semi-colons:

```
&dimension=qrur9Dvnyt5:GT:5:LT:15
```

The available operators are listed below.

Filter operators

Operator	Description
EQ	Equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to
NE	Not equal to
LIKE	Like (free text match)
IN	Equal to one of multiple values separated by ";"

Response formats

The default response representation format is JSON. The requests must be using the HTTP *GET* method. The following response formats are supported.

- json (application/json)
- jsonp (application/javascript)
- xls (application/vnd.ms-excel)

As an example, to get a response in Excel format you can use a file extension in the request URL like this:

```
/api/33/analytics/events/query/eBAyeGv0exc.xls?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu&dimension=oZg33kd9taw&dimension=qrur9Dvnyt5
```

You can set the hierarchyMeta query parameter to true in order to include names of all ancestor organisation units in the meta-section of the response:

```
/api/33/analytics/events/query/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:YuQRtpLP10I&dimension=qrur9Dvnyt5:EQ:50&hierarchyMeta=true
```

The default response JSON format will look similar to this:

```
{
    "headers": [
        {
            "name": "psi",
            "column": "Event",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "ps",
            "column": "Program stage",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "eventdate",
            "column": "Event date",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "coordinates",
            "column": "Coordinates",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "ouname",
            "column": "Organisation unit name",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "oucode",
            "column": "Organisation unit code",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "ou",
            "column": "Organisation unit",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
```

```
            "name": "oZg33kd9taw",
            "column": "Gender",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        },
        {
            "name": "qrur9Dvnyt5",
            "column": "Age",
            "type": "java.lang.String",
            "hidden": false,
            "meta": false
        }
    ],
    "metaData": {
        "names": {
            "qrur9Dvnyt5": "Age",
            "eBAyeGv0exc": "Inpatient morbidity and mortality",
            "ImspTQPwCqd": "Sierra Leone",
            "O6uvpzGd5pu": "Bo",
            "YuQRtpLP10I": "Badjia",
            "oZg33kd9taw": "Gender"
        },
        "ouHierarchy": {
            "YuQRtpLP10I": "/ImspTQPwCqd/O6uvpzGd5pu"
        }
    },
    "width": 8,
    "height": 4,
    "rows": [
        [
            "yx9IDINf82o",
            "Zj7UnCAulEk",
            "2016-08-05",
            "[5.12, 1.23]",
            "Ngelehun",
            "OU_559",
            "YuQRtpLP10I",
            "Female",
            "50"
        ],
        [
            "IPNa7AsCyFt",
            "Zj7UnCAulEk",
            "2016-06-12",
            "[5.22, 1.43]",
            "Ngelehun",
            "OU_559",
            "YuQRtpLP10I",
            "Female",
            "50"
        ],
        [
            "ZY9JL9dkhD2",
            "Zj7UnCAulEk",
            "2016-06-15",
            "[5.42, 1.33]",
            "Ngelehun",
            "OU_559",
            "YuQRtpLP10I",
            "Female",
            "50"
        ],
```

```
        [
            "MYvh4WAUdWt",
            "Zj7UnCAulEk",
            "2016-06-16",
            "[5.32, 1.53]",
            "Ngelehun",
            "OU_559",
            "YuQRtpLP10I",
            "Female",
            "50"
        ]
    ]
}
```

The *headers* section of the response describes the content of the query result. The event unique identifier, the program stage identifier, the event date, the organisation unit name, the organisation unit code and the organisation unit identifier appear as the first six dimensions in the response and will always be present. Next comes the data elements, person attributes and person identifiers which were specified as dimensions in the request, in this case, the "Gender" and "Age" data element dimensions. The header section contains the identifier of the dimension item in the "name" property and a readable dimension description in the "column" property.

The *metaData* section, *ou* object contains the identifiers of all organisation units present in the response mapped to a string representing the hierarchy. This hierarchy string lists the identifiers of the ancestors (parents) of the organisation unit starting from the root. The *names* object contains the identifiers of all items in the response mapped to their names.

The *rows* section contains the events produced by the query. Each row represents exactly one event.

In order to have the event analytics resource generate the data in the shape of a ready-made table, you can provide *rows* and *columns* parameters with requested dimension identifiers separated by semi-colons as values to indicate which ones to use as table columns and rows. Instead of generating a plain, normalized data source, the event analytics resource will now generate the data in table layout. The column and rows dimensions must be present as a data dimension in the query (not a filter). Such a request can look like this:

```
/api/33/analytics.html+css?dimension=dx:cYeuwXTCPkU;fbfJHSPpUQD&dimension=pe:WEEKS_THIS_YEAR
  &filter=ou:ImspTQPwCqd&displayProperty=SHORTNAME&columns=dx&rows=pe
```

Event aggregate analytics

The `/analytics/events/aggregate` resource lets you retrieve *aggregated numbers* of events captured in DHIS2. This resource lets you retrieve aggregate data based on a program and optionally a program stage, and lets you filter on any event dimension.

```
/api/33/analytics/events/aggregate
```

The events aggregate resource does not return the event information itself, rather the aggregate numbers of events matching the request query. Event dimensions include data elements, person attributes, person identifiers, periods and organisation units. Aggregate event queries should be on the format described below.

```
/api/33/analytics/events/aggregate/<program-id>?startDate=yyyy-MM-dd&endDate=yyyy-MM-dd
  &dimension=ou:<ou-id>;<ou-id>&dimension=<item-id>&dimension=<item-id>:<operator>:<filter>
```

For example, to retrieve aggregate numbers for events from the "Inpatient morbidity and mortality" program between January and October 2016, where the "Gender" and "Age" data elements are included, the "Age" dimension item is filtered on "18" and the "Gender" item is filtered on "Female", you can use the following query:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu&dimension=oZg33kd9taw:EQ:Female&dimension=qrur9Dvnyt5:GT:50
```

To retrieve data for fixed and relative periods instead of start and end date, in this case, May 2016 and last 12 months, and the organisation unit associated with the current user, you can use the following query:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc?dimension=pe:201605;LAST_12_MONTHS
  &dimension=ou:USER_ORGUNIT;fdc6uOvgo7ji&dimension=oZg33kd9taw
```

In order to specify "Female" as a filter for "Gender" for the data response, meaning "Gender" will not be part of the response but will filter the aggregate numbers in it, you can use the following syntax:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc?dimension=pe:2016;
  &dimension=ou:O6uvpzGd5pu&filter=oZg33kd9taw:EQ:Female
```

To specify the "Bo" organisation unit and the period "2016" as filters, and the "Mode of discharge" and Gender" as dimensions, where "Gender" is filtered on the "Male" item, you can use a query like this:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc?filter=pe:2016&filter=ou:O6uvpzGd5pu
  &dimension=fWIAEtYVEGk&dimension=oZg33kd9taw:EQ:Male
```

To create a "Top 3 report" for *Mode of discharge* you can use the limit and sortOrder query parameters similar to this:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc?filter=pe:2016&filter=ou:O6uvpzGd5pu
  &dimension=fWIAEtYVEGk&limit=3&sortOrder=DESC
```

To specify a value dimension with a corresponding aggregation type you can use the value and aggregationType query parameters. Specifying a value dimension will make the analytics engine return aggregate values for the values of that dimension in the response as opposed to counts of events.

```
/api/33/analytics/events/aggregate/eBAyeGv0exc.json?stage=Zj7UnCAulEk
  &dimension=ou:ImspTQPwCqd&dimension=pe:LAST_12_MONTHS&dimension=fWIAEtYVEGk
  &value=qrur9Dvnyt5&aggregationType=AVERAGE
```

To base event analytics aggregation on a specific data element or attribute of value type date or date time you can use the `timeField` parameter:

```
/api/33/analytics/events/aggregate/IpHINAT79UW.json?dimension=ou:ImspTQPwCqd
  &dimension=pe:LAST_12_MONTHS&dimension=cejWyOfXge6&stage=A03MvHHogjR
  &timeField=ENROLLMENT_DATE
```

To base event analytics aggregation on a specific data element or attribute of value type organisation unit you can use the `orgUnitField` parameter:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc.json?dimension=ou:ImspTQPwCqd
  &dimension=pe:THIS_YEAR&dimension=oZg33kd9taw&stage=Zj7UnCAulEk
  &orgUnitField=S33cRBsnXPo
```

Ranges / legend sets

For aggregate queries, you can specify a range / legend set for numeric data element and attribute dimensions. The purpose is to group the numeric values into ranges. As an example, instead of generating data for an "Age" data element for distinct years, you can group the information into age groups. To achieve this, the data element or attribute must be associated with the legend set. The format is described below:

```
?dimension=<item-id>-<legend-set-id>
```

An example looks like this:

```
/api/33/analytics/events/aggregate/eBAyeGv0exc.json?stage=Zj7UnCAulEk
  &dimension=qrur9Dvnyt5-Yf6UHoPkdS6&dimension=ou:ImspTQPwCqd&dimension=pe:LAST_MONTH
```

Response formats

The default response representation format is JSON. The requests must be using the HTTP *GET* method. The response will look similar to this:

```
{
    "headers": [
        {
            "name": "oZg33kd9taw",
            "column": "Gender",
            "type": "java.lang.String",
            "meta": false
        },
        {
            "name": "qrur9Dvnyt5",
            "column": "Age",
            "type": "java.lang.String",
            "meta": false
        },
        {
            "name": "pe",
            "column": "Period",
            "type": "java.lang.String",
            "meta": false
```

```
        },
        {
            "name": "ou",
            "column": "Organisation unit",
            "type": "java.lang.String",
            "meta": false
        },
        {
            "name": "value",
            "column": "Value",
            "type": "java.lang.String",
            "meta": false
        }
    ],
    "metaData": {
        "names": {
            "eBAyeGv0exc": "Inpatient morbidity and mortality"
        }
    },
    "width": 5,
    "height": 39,
    "rows": [
        ["Female", "95", "201605", "O6uvpzGd5pu", "2"],
        ["Female", "63", "201605", "O6uvpzGd5pu", "2"],
        ["Female", "67", "201605", "O6uvpzGd5pu", "1"],
        ["Female", "71", "201605", "O6uvpzGd5pu", "1"],
        ["Female", "75", "201605", "O6uvpzGd5pu", "14"],
        ["Female", "73", "201605", "O6uvpzGd5pu", "5"]
    ]
}
```


Note that the max limit for rows to return in a single response is 10 000. If the query produces more than the max limit, a *409 Conflict* status code will be returned.

Event clustering analytics

The *analytics/events/cluster* resource provides clustered geospatial event data. A request looks like this:

```
/api/33/analytics/events/cluster/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:LEVEL-2&clusterSize=100000
  &bbox=-13.2682125,7.3721619,-10.4261178,9.904012&includeClusterPoints=false
```

The cluster response provides the count of underlying points, the center point and extent of each cluster. If the `includeClusterPoints` query parameter is set to true, a comma-separated string with the identifiers of the underlying events is included. A sample response looks like this:


```
{
    "headers": [
        {
            "name": "count",
            "column": "Count",
            "type": "java.lang.Long",
            "meta": false
        },
        {
            "name": "center",
            "column": "Center",
            "type": "java.lang.String",
```

```
            "meta": false
        },
        {
            "name": "extent",
            "column": "Extent",
            "type": "java.lang.String",
            "meta": false
        },
        {
            "name": "points",
            "column": "Points",
            "type": "java.lang.String",
            "meta": false
        }
    ],
    "width": 3,
    "height": 4,
    "rows": [
        [
            "3",
            "POINT(-13.15818 8.47567)",
            "BOX(-13.26821 8.4St7215,-13.08711 8.47807)",
            ""
        ],
        [
            "9",
            "POINT(-13.11184 8.66424)",
            "BOX(-13.24982 8.51961,-13.05816 8.87696)",
            ""
        ],
        [
            "1",
            "POINT(-12.46144 7.50597)",
            "BOX(-12.46144 7.50597,-12.46144 7.50597)",
            ""
        ],
        [
            "7",
            "POINT(-12.47964 8.21533)",
            "BOX(-12.91769 7.66775,-12.21011 8.49713)",
            ""
        ]
    ]
}
```


Event count and extent analytics

The *analytics/events/count* resource is suitable for geometry-related requests for retrieving the count and extent (bounding box) of events for a specific query. The query syntax is equal to the *events/query* resource. A request looks like this:

```
/api/33/analytics/events/count/eBAyeGv0exc?startDate=2016-01-01
  &endDate=2016-10-31&dimension=ou:O6uvpzGd5pu
```

The response will provide the count and extent in JSON format:


```
{
    "extent": "BOX(-13.2682125910096 7.38679562779441,-10.4261178860988 9.90401290212795)",
```

```
    "count": 59
}
```

Constraints and validation

There are several constraints to the input parameters you can provide to the event analytics resource. If any of the constraints are violated, the API will return a *409 Conflict* response and a response message looking similar to this:

```
{
    "httpStatus": "Conflict",
    "httpStatusCode": 409,
    "status": "ERROR",
    "message": "At least one organisation unit must be specified",
    "errorCode": "E7200"
}
```

The possible validation errors for the event analytics API are described in the table below.

Error code	Message
E7200	At least one organisation unit must be specified
E7201	Dimensions cannot be specified more than once
E7202	Query items cannot be specified more than once
E7203	Value dimension cannot also be specified as an item or item filter
E7204	Value dimension or aggregate data must be specified when aggregation type is specified
E7205	Start and end date or at least one period must be specified
E7206	Start date is after end date
E7207	Page number must be a positive number
E7208	Page size must be zero or a positive number
E7209	Limit is larger than max limit
E7210	Time field is invalid
E7211	Org unit field is invalid
E7212	Cluster size must be a positive number
E7213	Bbox is invalid, must be on format: 'min-lng,min-lat,max-lng,max-lat'
E7214	Cluster field must be specified when bbox or cluster size are specified
E7215	Query item cannot specify both legend set and option set
E7216	Query item must be aggregateable when used in aggregate query
E7217	User is not allowed to view event analytics data
E7218	Spatial database support is not enabled
E7219	Data element must be of value type coordinate in order to be used as coordinate field

Error code	Message
E7220	Attribute must be of value type coordinate to in order to be used as coordinate field
E7221	Coordinate field is invalid
E7222	Query item or filter is invalid
E7223	Value does not refer to a data element or attribute which are numeric and part of the program
E7224	Item identifier does not reference any data element, attribute or indicator part of the program
E7225	Program stage is mandatory for data element dimensions in enrollment analytics queries
E7226	Dimension is not a valid query item
E7227	Relationship entity type not supported

Enrollment analytics

The enrollment analytics API lets you access aggregated event data and query *enrollments with their event data* captured in DHIS2. This resource lets you retrieve data for a program based on program stages and data elements - in addition to tracked entity attributes. When querying event data for a specific programstages within each enrollment, the data element values for each program stage will be returned as one row in the response from the api. If querying a data element in a program stage that is repeatable, the newest data element value will be used for that data element in the api response.

Dimensions and items

Enrollment dimensions include data elements, attributes, organisation units and periods. The query analytics resource will simply return enrollments matching a set of criteria and does not perform any aggregation.

Enrollment dimensions

Dimension	Dimension id	Description
Data elements in program stages	<program stage id>.<data element id>	Data element identifiers must include the program stage when querying data for enrollments. dimension=edqlbukwRfQ.vANAXwtLwcT
Attributes	<id>	Attribute identifiers
Periods	pe	ISO periods and relative periods, see "date and period format"
Organisation units	ou	Organisation unit identifiers and keywords USER_ORGUNIT, USER_ORGUNIT_CHILDREN, USER_ORGUNIT_GRANDCHILDREN, LEVEL-<level> and OU_GROUP-<group-id>

Enrollment query analytics

The *analytics/enrollments/query* resource lets you query for captured enrollments. This resource does not perform any aggregation, rather it lets you query and filter for information about enrollments.

```
/api/33/analytics/enrollments/query
```

You can specify any number of dimensions and any number of filters in a query. Dimension item identifiers can refer to any of the data elements in program stages, tracked entity attributes, fixed and relative periods and organisation units. Dimensions can optionally have a query operator and a filter. Enrollment queries should be on the format described below.

```
/api/33/analytics/enrollments/query/<program-id>?startDate=yyyy-MM-dd&endDate=yyyy-MM-dd
  &dimension=ou:<ou-id>;<ou-id>&dimension=<item-id>&dimension=<item-id>:<operator>:<filter>
```

For example, to retrieve enrollments in the from the "Antenatal care" program from January 2019, where the "First name" is picked up from attributes, "Chronic conditions" and "Smoking" data elements are included from the first program stage, and "Hemoglobin value" from the following program stage, and only women that have "Cronic conditions" would be included, you can use the following query:

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.json?dimension=ou:ImspTQPwCqd
  &dimension=w75KJ2mc4zz&dimension=WZbXY0S00lP.de0FEHSIoxh:eq:1&dimension=w75KJ2mc4zz
  &dimension=WZbXY0S00lP.sWoqcoByYmD&dimension=edqlbukwRfQ.vANAXwtLwcT
  &startDate=2019-01-01&endDate=2019-01-31
```

To retrieve enrollments in the from the "Antenatal care" program from last month (relative to the point in time the query is executed), where the "Chronic conditions" and "Smoking" data elements are included from the first program stage, and "Hemoglobin value" from the followup program stage, only including smoking women with hemoglobin less than 20:

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.json?dimension=ou:ImspTQPwCqd
  &dimension=WZbXY0S00lP.de0FEHSIoxh&dimension=w75KJ2mc4zz
  &dimension=WZbXY0S00lP.sWoqcoByYmD:eq:1&dimension=edqlbukwRfQ.vANAXwtLwcT:lt:20
  &dimension=pe:LAST_MONTH
```

Sorting can be applied to the query for the enrollment and incident dates of the enrollment:

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.xls?dimension=ou:ImspTQPwCqd
  &columns=w75KJ2mc4zz&dimension=WZbXY0S00lP.sWoqcoByYmD&dimension=pe:LAST_MONTH
  &stage=WZbXY0S00lP&pageSize=10&page=1&asc=ENROLLMENTDATE&ouMode=DESCENDANTS
```

Paging can be applied to the query by specifying the page number and the page size parameters. If page number is specified but page size is not, a page size of 50 will be used. If page size is specified but page number is not, a page number of 1 will be used. To get the second page of the response with a page size of 10 you can use a query like this:

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.json?dimension=ou:ImspTQPwCqd
  &dimension=WZbXY0S00lP.de0FEHSIoxh&dimension=w75KJ2mc4zz&dimension=pe:LAST_MONTH
  &dimension=WZbXY0S00lP.sWoqcoByYmD&pageSize=10&page=2
```

Filtering

Filters can be applied to data elements, person attributes and person identifiers. The filtering is done through the query parameter value on the following format:

```
&dimension=<item-id>:<operator>:<filter-value>
```

As an example, you can filter the "Weight" data element for values greater than 2000 and lower than 4000 like this:

```
&dimension=WZbXY0S00lP.UXz7xuGCEhU:GT:2000&dimension=WZbXY0S00lP.UXz7xuGCEhU:LT:4000
```

You can filter the "Age" attribute for multiple, specific ages using the IN operator like this:

```
&dimension=qrur9Dvnyt5:IN:18;19;20
```

You can specify multiple filters for a given item by repeating the operator and filter components, all separated with semi-colons:

```
&dimension=qrur9Dvnyt5:GT:5:LT:15
```

The available operators are listed below.

Filter operators

Operator	Description
EQ	Equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to
NE	Not equal to
LIKE	Like (free text match)
IN	Equal to one of multiple values separated by ";"

Request query parameters

The analytics enrollment query API lets you specify a range of query parameters.

Query parameters for enrollment query endpoint

Query parameter	Required	Description	Options (default first)
program	Yes	Program identifier.	Any program identifier
startDate	No	Start date for enrollments.	Date in yyyy-MM-dd format
endDate	No	End date for enrollments.	Date in yyyy-MM-dd format
dimension	Yes	Dimension identifier including data elements, attributes, program indicators, periods, organisation units and organisation unit group sets. Parameter can be repeated any number of times. Item filters can be applied to a dimension on the format <item-id>:<operator>:<filter>. Filter values are case-insensitive.	Operators can be EQ | GT | GE | LT | LE | NE | LIKE | IN
filter	No	Dimension identifier including data elements, attributes, periods, organisation units and organisation unit group sets. Parameter can be repeated any number of times. Item filters can be applied to a dimension on the format <item-id>:<operator>:<filter>. Filter values are case-insensitive.	
programStatus	No	Specify enrollment status of enrollments to include.	ACTIVE | COMPLETED | CANCELLED
relativePeriodDate	string	No	Date identifier e.g: "2016-01-01". Overrides the start date of the relative period
ouMode	No	The mode of selecting organisation units. Default is DESCENDANTS, meaning all sub units in the hierarchy. CHILDREN refers to immediate children in the hierarchy; SELECTED refers to the selected organisation units only.	DESCENDANTS, CHILDREN, SELECTED
asc	No	Dimensions to be sorted ascending, can reference enrollment date, incident date, org unit name and code.	ENROLLMENTDATE | INCIDENTDATE| OUNAME | OUCODE
desc	No	Dimensions to be sorted descending, can reference enrollment date, incident date, org unit name and code.	ENROLLMENTDATE | INCIDENTDATE| OUNAME | OUCODE
hierarchyMeta	No	Include names of organisation unit ancestors and hierarchy paths of organisation units in the metadata.	false | true

Query parameter	Required	Description	Options (default first)
coordinatesOnly	No	Whether to only return enrollments which have coordinates.	false | true
page	No	The page number. Default page is 1.	Numeric positive value
pageSize	No	The page size. Default size is 50 items per page.	Numeric zero or positive value

Response formats

The default response representation format is JSON. The requests must be using the HTTP *GET* method. The following response formats are supported.

- json (application/json)
- xml (application/xml)
- xls (application/vnd.ms-excel)
- csv (application/csv)
- html (text/html)
- html+css (text/html)

As an example, to get a response in Excel format you can use a file extension in the request URL like this:

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.xls?dimension=ou:ImspTQPwCqd
  &dimension=WZbXY0S00lP.de0FEHSIoxh&columns=w75KJ2mc4zz
  &dimension=WZbXY0S00lP.sWoqcoByYmD&dimension=pe:LAST_MONTH&stage=WZbXY0S00lP
  &pageSize=10&page=1&asc=ENROLLMENTDATE&ouMode=DESCENDANTS
```

The default response JSON format will look similar to this:

```
{
    "headers": [
        {
            "name": "pi",
            "column": "Enrollment",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "tei",
            "column": "Tracked entity instance",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "enrollmentdate",
            "column": "Enrollment date",
            "valueType": "DATE",
            "type": "java.util.Date",
            "hidden": false,
            "meta": true
```

```
        },
        {
            "name": "incidentdate",
            "column": "Incident date",
            "valueType": "DATE",
            "type": "java.util.Date",
            "hidden": false,
            "meta": true
        },
        {
            "name": "geometry",
            "column": "Geometry",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "longitude",
            "column": "Longitude",
            "valueType": "NUMBER",
            "type": "java.lang.Double",
            "hidden": false,
            "meta": true
        },
        {
            "name": "latitude",
            "column": "Latitude",
            "valueType": "NUMBER",
            "type": "java.lang.Double",
            "hidden": false,
            "meta": true
        },
        {
            "name": "ouname",
            "column": "Organisation unit name",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "oucode",
            "column": "Organisation unit code",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "ou",
            "column": "Organisation unit",
            "valueType": "TEXT",
            "type": "java.lang.String",
            "hidden": false,
            "meta": true
        },
        {
            "name": "de0FEHSIoxh",
            "column": "WHOMCH Chronic conditions",
            "valueType": "BOOLEAN",
            "type": "java.lang.Boolean",
            "hidden": false,
```

```
            "meta": true
        },
        {
            "name": "sWoqcoByYmD",
            "column": "WHOMCH Smoking",
            "valueType": "BOOLEAN",
            "type": "java.lang.Boolean",
            "hidden": false,
            "meta": true
        }
    ],
    "metaData": {
        "pager": {
            "page": 2,
            "total": 163,
            "pageSize": 4,
            "pageCount": 41
        },
        "items": {
            "ImspTQPwCqd": {
                "name": "Sierra Leone"
            },
            "PFDfvmGpsR3": {
                "name": "Care at birth"
            },
            "bbKtnxRZKEP": {
                "name": "Postpartum care visit"
            },
            "ou": {
                "name": "Organisation unit"
            },
            "PUZaKR0Jh2k": {
                "name": "Previous deliveries"
            },
            "edqlbukwRfQ": {
                "name": "Antenatal care visit"
            },
            "WZbXY0S00lP": {
                "name": "First antenatal care visit"
            },
            "sWoqcoByYmD": {
                "name": "WHOMCH Smoking"
            },
            "WSGAb5XwJ3Y": {
                "name": "WHO RMNCH Tracker"
            },
            "de0FEHSIoxh": {
                "name": "WHOMCH Chronic conditions"
            }
        },
        "dimensions": {
            "pe": [],
            "ou": ["ImspTQPwCqd"],
            "sWoqcoByYmD": [],
            "de0FEHSIoxh": []
        }
    },
    "width": 12,
    "rows": [
        [
            "A0cP533hIQv",
            "to8G9jAprnx",
            "2019-02-02 12:05:00.0",
```

```
            "2019-02-02 12:05:00.0",
            "",
            "0.0",
            "0.0",
            "Tonkomba MCHP",
            "OU_193264",
            "xIMxph4NMP1",
            "0",
            "1"
        ],
        [
            "ZqiUn2uXmBi",
            "SJtv0WzoYki",
            "2019-02-02 12:05:00.0",
            "2019-02-02 12:05:00.0",
            "",
            "0.0",
            "0.0",
            "Mawoma MCHP",
            "OU_254973",
            "Srnpwq8jKbp",
            "0",
            "0"
        ],
        [
            "lE747mUAtbz",
            "PGzTv2A1xzn",
            "2019-02-02 12:05:00.0",
            "2019-02-02 12:05:00.0",
            "",
            "0.0",
            "0.0",
            "Kunsho CHP",
            "OU_193254",
            "tdhB1JXYBx2",
            "",
            "0"
        ],
        [
            "nmcqu9QF8ow",
            "pav3tGLjYuq",
            "2019-02-03 12:05:00.0",
            "2019-02-03 12:05:00.0",
            "",
            "0.0",
            "0.0",
            "Korbu MCHP",
            "OU_678893",
            "m73lWmo5BDG",
            "",
            "1"
        ]
    ],
    "height": 4
}
```


The *headers* section of the response describes the content of the query result. The enrollment unique identifier, the tracked entity instance identifier, the enrollment date, the incident date, geometry, latitude, longitude, the organisation unit name and the organisation unit code appear as the first dimensions in the response and will always be present. Next comes the data elements, and tracked entity attributes which were specified as dimensions in the request, in this case, the

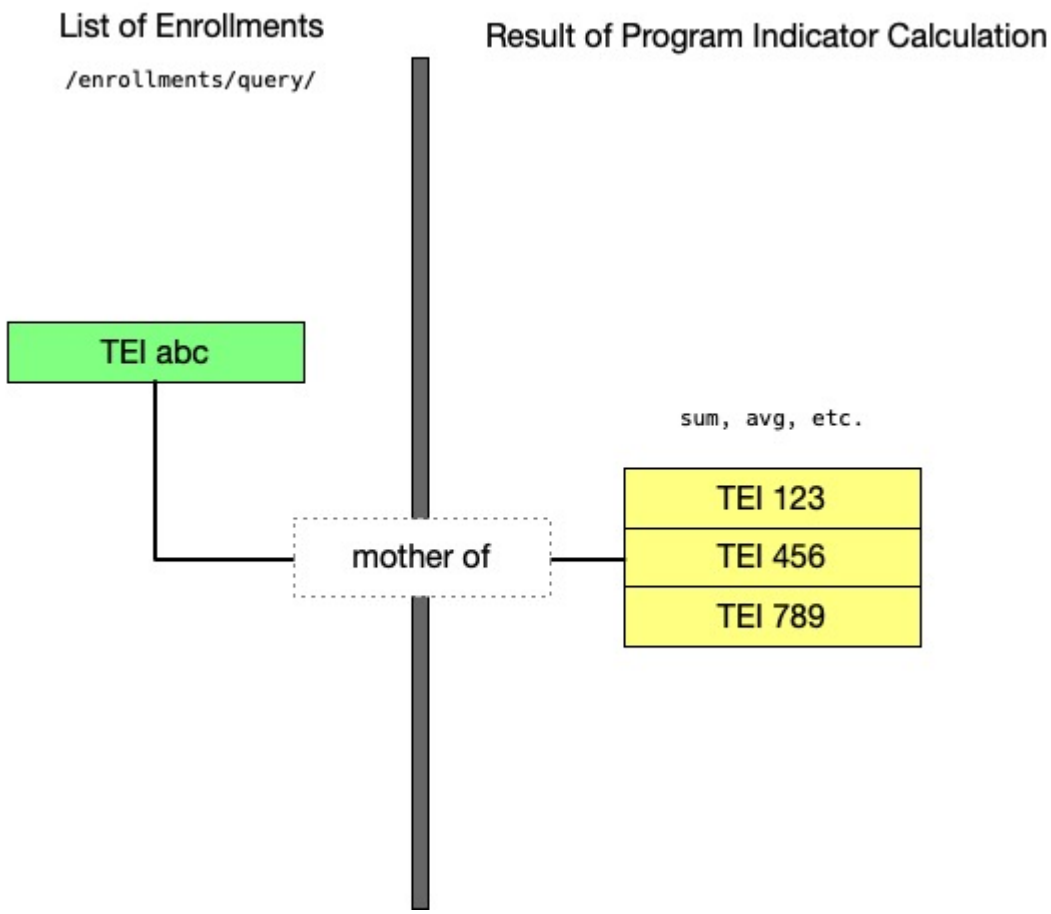
"WHOMCH Chronic conditions" and "WHOMCH smoking" data element dimensions. The header section contains the identifier of the dimension item in the "name" property and a readable dimension description in the "column" property.

The *metaData* section, *ou* object contains the identifiers of all organisation units present in the response mapped to a string representing the hierarchy. This hierarchy string lists the identifiers of the ancestors (parents) of the organisation unit starting from the root. The *names* object contains the identifiers of all items in the response mapped to their names.

The *rows* section contains the enrollments produced by the query. Each row represents exactly one enrollment.

Support of analytics across tracked entity instance relationships with program indicators

The non-aggregation enrollment analytics API also supports linking Program Indicators to Relationship Types, in order to show the result of a calculation of a specific Program Indicator applied to the related entities of the listed Tracked Entity Instance.



For the Program Indicator/Relationship Type link to work, the `/api/33/analytics/enrollments/query` API requires an additional dimension which must include the chosen Relationship Type UID and the chosen Program Indicator UID:

```
/api/33/analytics/enrollments/query/<program-id>
  ?dimension=<relationshiptype-id>.<programindicator-id>
```

For example, to retrieve a list of enrollments from the "WHO RMNCH Tracker" program for January 2019 and display the count of Malaria Cases linked to that Enrollment by "Malaria case linked to person" type of relationship, you can use the following query

```
/api/33/analytics/enrollments/query/WSGAb5XwJ3Y.json?dimension=mxZDvSZYxlw.nFICjJluo74
  &startDate=2019-01-01&endDate=2019-01-31
```

The API supports using program indicators which are not associated to the "main" program (that is the program ID specified after `/query/`).

Org unit analytics

The org unit analytics API provides statistics on org units classified by org unit group sets, i.e. counts of org units per org unit group within org unit group sets.

```
GET /api/orgUnitAnalytics?ou=<org-unit-id>&ougs=<org-unit-group-set-id>
```

The API requires at least one organisation unit and at least one organisation unit group set. Multiple org units and group sets can be provided separated by a semicolon.

Request query parameters

The org unit analytics resource lets you specify a range of query parameters:

Org unit analytics query parameters

Property	Description	Required
ou	Org unit identifiers, potentially separated by a semicolon.	Yes
ougs	Org unit group set identifiers, potentially separated by a semicolon.	Yes
columns	Org unit group set identifiers, potentially separated by a semicolon. Defines which group sets are rendered as columns in a table layout.	No

The response will contain a column for the parent org unit, columns for each org unit group set part of the request and a column for the count. The statistics include the count of org units which are part of the sub-hierarchy of the org units specified in the request. The response contains a metadata section which specifies the name of each org unit and org unit group part of the response referenced by their identifiers.

The default response is normalized with a single `count` column. The response can be rendered in a table layout by specifying at least one org unit group set using the `columns` query parameter.

Response formats

The org unit analytics endpoint supports the following representation formats:

- json (application/json)
- csv (application/csv)
- xls (application/vnd.ms-excel)
- pdf (application/pdf)

Examples

To fetch org unit analytics for an org unit and org unit group set:

```
GET /api/orgUnitAnalytics?ou=lc3eMKXaEfw&ougs=J5jldMd8OHv
```

To fetch org unit analytics data for two org units and two org unit group sets:

```
GET /api/orgUnitAnalytics?ou=lc3eMKXaEfw;PMa2VCrupOd&ougs=J5jldMd8OHv;Bpx0589u8y0
```

To fetch org unit analytics data in table mode with one group set rendered as columns:

```
GET /api/orgUnitAnalytics?ou=fdc6uOvgoji;jUb8gELQApl;lc3eMKXaEfw;PMa2VCrupOd
  &ougs=J5jldMd8OHv&columns=J5jldMd8OHv
```

Constraints and validation

The possible validation errors specifically for the org unit analytics API are described in the table below. Some errors specified for the aggregate analytics API are also relevant.

Error code	Message
E7300	At least one organisation unit must be specified
E7301	At least one organisation unit group set must be specified

Data set report

Data set reports can be generated through the web api using the `/dataSetReport` resource. This resource generates reports on data set and returns the result in the form of an HTML table.

```
/api/33/dataSetReport
```

Request query parameters

The request supports the following parameters:

Data set report query parameters

Parameter	Description	Type	Required
ds	Data set to create the report from.	Data set UID	Yes
pe	Period to create the report from.	ISO String	Yes
ou	Organisation unit to create the report from.	Organisation unit UID	Yes
filter	Filters to be used as filters for the report. Can be repeated any number of times. Follows the analytics API syntax.	One or more UIDs	No
selectedUnitOnly	Whether to use captured data only or aggregated data.	Boolean	No

The data set report resource accepts `GET` requests only. The response content type is `application/json` and returns data in a grid. This endpoint works for all types of data sets, including default, section and custom forms.

An example request to retrieve a report for a data set and org unit for 2018 looks like this:

```
GET /api/33/dataSetReport?ds=BfMAe6Itzgt&pe=201810&ou=ImspTQPwCqd&selectedUnitOnly=false
```

To get a data set report with a filter you can use the `filter` parameter. In this case, the filter is based on an org unit group set and two org unit groups:

```
GET /api/33/dataSetReport?ds=BfMAe6Itzgt&pe=201810&ou=ImspTQPwCqd
  &filter=J5jldMd8OHv:RXL3lPSK8oG;tDZVQ1WtwpA
```

Response formats

The data set report endpoint supports output in the following formats. You can retrieve a specific endpoint using the file extension or `Accept` HTTP header.

- json (application/json)
- pdf (application/pdf)
- xls (application/vnd.ms-excel)

Custom forms

A dedicated endpoint is available for data sets with custom HTML forms. This endpoint returns the HTML form content with content type `text/html` with data inserted into it. Note that you can use the general data set report endpoint also for data sets with custom forms; however, that will return the report in JSON format as a grid. This endpoint only works for data sets with custom HTML forms.

```
GET /api/33/dataSetReport/custom
```

The syntax for this endpoint is otherwise equal to the general data set report endpoint. To retrieve a custom HTML data set report you can issue a request like this:

```
GET /api/33/dataSetReport/custom?ds=lyLU2wR22tC&pe=201810&ou=ImspTQPwCqd
```

Push Analysis

The push analysis API includes endpoints for previewing a push analysis report for the logged in user and manually triggering the system to generate and send push analysis reports, in addition to the normal CRUD operations. When using the create and update endpoints for push analysis, the push analysis will be scheduled to run based on the properties of the push analysis. When deleting or updating a push analysis to be disabled, the job will also be stopped from running in the future.

To get an HTML preview of an existing push analysis, you can do a GET request to the following endpoint:

```
/api/33/pushAnalysis/<id>/render
```

To manually trigger a push analysis job, you can do a POST request to this endpoint:

```
/api/33/pushAnalysis/<id>/run
```

A push analysis consists of the following properties, where some are required to automatically run push analysis jobs:

Push analysis properties

Property	Description	Type	Required
dashboard	Dashboard on which reports are based	Dashboard UID	Yes
message	Appears after title in reports	String	No
recipientUserGroups	A set of user groups who should receive the reports	One or more user Group UID	No. Scheduled jobs without any recipient will be skipped.
enabled	Indicated whether this push analysis should be scheduled or not. False by default.	Boolean	Yes. Must be true to be scheduled.
schedulingFrequency	The frequency of which reports should be scheduled.	"DAILY", "WEEKLY", "MONTHLY"	No. Push analysis without a frequency will not be scheduled
schedulingDayOfFrequency	The day in the frequency the job should be scheduled.	Integer. Any value when frequency is "DAILY". 0-7 when frequency is "WEEKLY". 1-31 when frequency is "MONTHLY"	No. Push analysis without a valid day of frequency for the frequency set will not be scheduled.

Data usage analytics

The usage analytics API lets you access information about how people are using DHIS2 based on data analysis. When users access favorites, an event is recorded. The event consists of the user name, the UID of the favorite, when the event took place, and the type of event. The different types of events are listed in the table.

```
/api/33/dataStatistics
```

The usage analytics API lets you retrieve aggregated snapshots of usage analytics based on time intervals. The API captures user views (for example the number of times a chart or pivot table has been viewed by a user) and saved analysis favorites (for example favorite charts and pivot tables). DHIS2 will capture nightly snapshots which are then aggregated at request.

Request query parameters

The usage analytics (data statistics) API supports two operations:

- *POST:* creates a view event
- *GET:* retrieves aggregated statistics

Create view events (POST)

The usage analytics API lets you create event views. The dataStatisticsEventType parameter describes what type of item was viewed. The favorite parameter indicates the identifier of the relevant favorite.

URL that creates a new event view of charts:

```
POST /api/33/dataStatistics?eventType=CHART_VIEW&favorite=LW0O27b7TdD
```

A successful save operation returns an HTTP status code 201. The table below shows the supported types of events.

Supported event types

Key	Description
REPORT_TABLE_VIEW	Report table (pivot table) view
CHART_VIEW	Chart view
MAP_VIEW	Map view (GIS)
EVENT_REPORT_VIEW	Event report view
EVENT_CHART_VIEW	Event chart view
DASHBOARD_VIEW	Dashboard view
DATA_SET_REPORT_VIEW	Data set report view

Retrieve aggregated usage analytics report (GET)

The usage analytics (data statistics) API lets you specify certain query parameters when asking for an aggregated report.

Query parameters for aggregated usage analytics (data statistics)

Query parameter	Required	Description	Options
startDate	Yes	Start date for period	Date in yyyy-MM-dd format
endDate	Yes	End date for period	Date in yyyy-MM-dd format
interval	Yes	Type of interval to be aggregated	DAY, WEEK, MONTH, YEAR

The startDate and endDate parameters specify the period for which snapshots are to be used in the aggregation. You must format the dates as shown above. If no snapshots are saved in the specified period, an empty list is sent back. The parameter called interval specifies what type of aggregation will be done.

API query that creates a query for a monthly aggregation:

```
GET /api/33/dataStatistics?startDate=2014-01-02&endDate=2016-01-01&interval=MONTH
```

Retrieve top favorites

The usage analytics API lets you retrieve the top favorites used in DHIS2, and by user.

Query parameters for top favorites

Query parameter	Required	Description	Options
eventType	Yes	The data statistics event type	See above table
pageSize	No	Size of the list returned	For example 5, 10, 25. Default is 25
sortOrder	No	Descending or ascending	ASC or DESC. Default is DESC.
username	No	If specified, the response will only contain favorites by this user.	For example 'admin'

The API query can be used without a username, and will then find the top favorites of the system.

```
/api/33/dataStatistics/favorites?eventType=CHART_VIEW&pageSize=25&sortOrder=ASC
```

If the username is specified, the response will only contain the top favorites of that user.

```
/api/33/dataStatistics/favorites?eventType=CHART_VIEW&pageSize=25
  &sortOrder=ASC&username=admin
```

Response format

You can return the aggregated data in a usage analytics response in several representation formats. The default format is JSON. The available formats and content types are:

- json (application/json)
- xml (application/xml)
- html (text/html)

API query that requests a usage analytics response in XML format:

```
/api/33/dataStatistics.xml?startDate=2014-01-01&endDate=2016-01-01&interval=WEEK
```

You must retrieve the aggregated usage analytics response with the HTTP GET method. This allows you to link directly from Web pages and other HTTP-enabled clients to usage analytics responses. To do functional testing use the cURL library.

To get an usage analytics response in JSON format:

```
/api/33/dataStatistics?startDate=2016-02-01&endDate=2016-02-14&interval=WEEK
```

The JSON response looks like this:

```
[
    {
        "year": 2016,
        "week": 5,
        "mapViews": 2181,
        "chartViews": 2227,
        "reportTableViews": 5633,
        "eventReportViews": 6757,
        "eventChartViews": 9860,
        "dashboardViews": 10082,
        "totalViews": 46346,
        "averageViews": 468,
        "averageMapViews": 22,
        "averageChartViews": 22,
        "averageReportTableViews": 56,
        "averageEventReportViews": 68,
        "averageEventChartViews": 99,
        "averageDashboardViews": 101,
        "savedMaps": 1805,
        "savedCharts": 2205,
        "savedReportTables": 1995,
        "savedEventReports": 1679,
        "savedEventCharts": 1613,
        "savedDashboards": 0,
        "savedIndicators": 1831,
        "activeUsers": 99,
        "users": 969
    },
    {
        "year": 2016,
        "week": 6,
        "mapViews": 2018,
        "chartViews": 2267,
        "reportTableViews": 4714,
        "eventReportViews": 6697,
        "eventChartViews": 9511,
        "dashboardViews": 12181,
        "totalViews": 47746,
        "averageViews": 497,
        "averageMapViews": 21,
        "averageChartViews": 23,
        "averageReportTableViews": 49,
        "averageEventReportViews": 69,
        "averageEventChartViews": 99,
        "averageDashboardViews": 126,
        "savedMaps": 1643,
        "savedCharts": 1935,
        "savedReportTables": 1867,
        "savedEventReports": 1977,
        "savedEventCharts": 1714,
        "savedDashboards": 0,
        "savedIndicators": 1646,
        "activeUsers": 96,
        "users": 953
    }
]
```


Retrieve statistics for a favorite

You can retrieve the number of view for a specific favorite by using the *favorites* resource, where *{favorite-id}* should be substituted with the identifier of the favorite of interest:

```
/api/33/dataStatistics/favorites/{favorite-id}.json
```

The response will contain the number of views for the given favorite and look like this:

```
{
    "views": 3
}
```

Geospatial features

The *geoFeatures* resource lets you retrieve geospatial information from DHIS2. Geospatial features are stored together with organisation units. The syntax for retrieving features is identical to the syntax used for the organisation unit dimension for the analytics resource. It is recommended to read up on the analytics api resource before continuing to read this section. You must use the GET request type, and only JSON response format is supported.

As an example, to retrieve geo features for all organisation units at level 3 in the organisation unit hierarchy you can use a GET request with the following URL:

```
/api/33/geoFeatures.json?ou=ou:LEVEL-3
```

To retrieve geo features for organisation units at a level within the boundary of an organisation unit (e.g. at level 2) you can use this URL:

```
/api/33/geoFeatures.json?ou=ou:LEVEL-4;O6uvpzGd5pu
```

The semantics of the response properties are described in the following table.

Geo features response

Property	Description
id	Organisation unit / geo feature identifier
na	Organisation unit / geo feature name
hcd	Has coordinates down, indicating whether one or more children organisation units exist with coordinates (below in the hierarchy)
hcu	Has coordinates up, indicating whether the parent organisation unit has coordinates (above in the hierarchy)
le	Level of this organisation unit / geo feature.
pg	Parent graph, the graph of parent organisation unit identifiers up to the root in the hierarchy
pi	Parent identifier, the identifier of the parent of this organisation unit
pn	Parent name, the name of the parent of this organisation unit
ty	Geo feature type, 1 = point and 2 = polygon or multi-polygon
co	Coordinates of this geo feature

GeoJSON

To export GeoJSON, you can simply add *.geosjon* as an extension to the endpoint */api/organisationUnits*, or you can use the *Accept* header *application/json+geojson*.

Two parameters are supported: `level` (default is 1) and `parent` (default is root organisation units). Both can be included multiple times. Some examples:

Get all features at level 2 and 4:

```
/api/organisationUnits.geojson?level=2&level=4
```

Get all features at level 3 with a boundary organisation unit:

```
/api/organisationUnits.geojson?parent=fdc6uOvgoji&level=3
```

Generating resource and analytics tables

DHIS2 features a set of generated database tables which are used as a basis for various system functionality. These tables can be executed immediately or scheduled to be executed at regular intervals through the user interface. They can also be generated through the Web API as explained in this section. This task is typically one for a system administrator and not consuming clients.

The resource tables are used internally by the DHIS2 application for various analysis functions. These tables are also valuable for users writing advanced SQL reports. They can be generated with a POST or PUT request to the following URL:

```
/api/33/resourceTables
```

The analytics tables are optimized for data aggregation and used currently in DHIS2 for the pivot table module. The analytics tables can be generated with a POST or PUT request to:

```
/api/33/resourceTables/analytics
```

Analytics tables optional query parameters

Query parameter	Options	Description
skipResourceTables	false | true	Skip generation of resource tables
skipAggregate	false | true	Skip generation of aggregate data and completeness data
skipEvents	false | true	Skip generation of event data
skipEnrollment	false | true	Skip generation of enrollment data
lastYears	integer	Number of last years of data to include

"Data Quality" and "Data Surveillance" can be run through the monitoring task, triggered with the following endpoint:

```
/api/33/resourceTables/monitoring
```

This task will analyse your validation rules, find any violations and persist them as validation results.

These requests will return immediately and initiate a server-side process.

Maintenance

To perform maintenance you can interact with the *maintenance* resource. You should use *POST* or *PUT* as a method for requests. The following methods are available.

Analytics tables clear will drop all analytics tables.

```
POST PUT /api/maintenance/analyticsTablesClear
```

Analytics table analyze will collects statistics about the contents of analytics tables in the database.

```
POST PUT /api/maintenance/analyticsTablesAnalyze
```

Expired invitations clear will remove all user account invitations which have expired.

```
POST PUT /api/maintenance/expiredInvitationsClear
```

Period pruning will remove periods which are not linked to any data values.

```
POST PUT /api/maintenance/periodPruning
```

Zero data value removal will delete zero data values linked to data elements where zero data is defined as not significant:

```
POST PUT /api/maintenance/zeroDataValueRemoval
```

Soft deleted data value removal will permanently delete soft deleted data values.

```
POST PUT /api/maintenance/softDeletedDataValueRemoval
```

Soft deleted program stage instance removal will permanently delete soft deleted events.

```
POST PUT /api/maintenance/softDeletedProgramStageInstanceRemoval
```

Soft deleted program instance removal will permanently delete soft deleted enrollments.

```
POST PUT /api/maintenance/softDeletedProgramInstanceRemoval
```

Soft deleted tracked entity instance removal will permanently delete soft deleted tracked entity instances.

```
POST PUT /api/maintenance/softDeletedTrackedEntityInstanceRemoval
```

Drop SQL views will drop all SQL views in the database. Note that it will not delete the DHIS2 SQL view entities.

```
POST PUT /api/maintenance/sqlViewsDrop
```

Create SQL views will recreate all SQL views in the database.

```
POST PUT /api/maintenance/sqlViewsCreate
```

Category option combo update will remove obsolete and generate missing category option combos for all category combinations.

```
POST PUT /api/maintenance/categoryOptionComboUpdate
```

It is also possible to update category option combos for a single category combo using the following endpoint.

```
POST PUT /api/maintenance/categoryOptionComboUpdate/categoryCombo/<category-combo-uid>
```

Cache clearing will clear the application Hibernate cache and the analytics partition caches.

```
POST PUT /api/maintenance/cacheClear
```

Org unit paths update will re-generate the organisation unit path property. This can be useful e.g. if you imported org units with SQL.

```
POST PUT /api/maintenance/ouPathsUpdate
```

Data pruning will remove complete data set registrations, data approvals, data value audits and data values, in this case for an organisation unit.

```
POST PUT /api/maintenance/dataPruning/organisationUnits/<org-unit-id>
```

Data pruning for data elements, which will remove data value audits and data values.

```
POST PUT /api/maintenance/dataPruning/dataElement/<data-element-uid>
```

Metadata validation will apply all metadata validation rules and return the result of the operation.

```
POST PUT /api/metadataValidation
```

App reload will refresh the DHIS2 managed cache of installed apps by reading from the file system.

```
POST PUT /api/appReload
```

Maintenance operations are supported in a batch style with a POST request to the api/maintenance resource where the operations are supplied as query parameters:

```
POST PUT /api/maintenance?analyticsTablesClear=true&expiredInvitationsClear=true
  &periodPruning=true&zeroDataValueRemoval=true&sqlViewsDrop=true&sqlViewsCreate=true
  &categoryOptionComboUpdate=true&cacheClear=true&ouPathsUpdate=true
```

System resource

The system resource provides you with convenient information and functions. The system resource can be found at */api/system*.

Generate identifiers

To generate valid, random DHIS2 identifiers you can do a GET request to this resource:

```
/api/33/system/id?limit=3
```

The *limit* query parameter is optional and indicates how many identifiers you want to be returned with the response. The default is to return one identifier. The response will contain a JSON object with an array named codes, similar to this:

```
{
    "codes": ["Y0moqFplrX4", "WI0VHXuWQuV", "BRJNBBpu4ki"]
}
```

The DHIS2 UID format has these requirements:

- 11 characters long.
- Alphanumeric characters only, ie. alphabetic or numeric characters (A-Za-z0-9).
- Start with an alphabetic character (A-Za-z).

View system information

To get information about the current system you can do a GET request to this URL:

```
/api/33/system/info
```

JSON and JSONP response formats are supported. The system info response currently includes the below properties.

```
{
    "contextPath": "http://yourdomain.com",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 Chrome/29.0.1547.62",
    "version": "2.13-SNAPSHOT",
    "revision": "11852",
    "buildTime": "2013-09-01T21:36:21.000+0000",
    "serverDate": "2013-09-02T12:35:54.311+0000",
```

```
    "environmentVariable": "DHIS2_HOME",
    "javaVersion": "1.7.0_06",
    "javaVendor": "Oracle Corporation",
    "javaIoTmpDir": "/tmp",
    "javaOpts": "-Xms600m -Xmx1500m -XX:PermSize=400m -XX:MaxPermSize=500m",
    "osName": "Linux",
    "osArchitecture": "amd64",
    "osVersion": "3.2.0-52-generic",
    "externalDirectory": "/home/dhis/config/dhis2",
    "databaseInfo": {
        "type": "PostgreSQL",
        "name": "dhis2",
        "user": "dhis",
        "spatialSupport": false
    },
    "memoryInfo": "Mem Total in JVM: 848 Free in JVM: 581 Max Limit: 1333",
    "cpuCores": 8
}
```


> **Note**
>
> If the user requesting this resource does not have full authority then only the first seven properties will be included, as other properties are considered sensitive information.

To get information about the system context only, i.e. `contextPath` and `userAgent`, you can make a GET request to the below URL. JSON and JSONP response formats are supported:

```
/api/33/system/context
```

Check if username and password combination is correct

To check if some user credentials (a username and password combination) is correct you can make a *GET* request to the following resource using *basic authentication*:

```
/api/33/system/ping
```

You can detect the outcome of the authentication by inspecting the *HTTP status code* of the response header. The meanings of the possible status codes are listed below. Note that this applies to Web API requests in general.

HTTP Status codes

HTTP Status code	Description	Outcome
200	OK	Authentication was successful
302	Found	No credentials were supplied with the request - no authentication took place
401	Unauthorized	The username and password combination was incorrect - authentication failed

View asynchronous task status

Tasks which often take a long time to complete can be performed asynchronously. After initiating an async task you can poll the status through the `system/tasks` resource by supplying the task category and the task identifier of interest.

When polling for the task status you need to authenticate as the same user which initiated the task. The following task categories are supported:

Task categories

Identifier	Description
ANALYTICS_TABLE	Generation of the analytics tables.
RESOURCE_TABLE	Generation of the resource tables.
MONITORING	Processing of data surveillance/monitoring validation rules.
DATAVALUE_IMPORT	Import of data values.
EVENT_IMPORT	Import of events.
ENROLLMENT_IMPORT	Import of enrollments.
TEI_IMPORT	Import of tracked entity instances.
METADATA_IMPORT	Import of metadata.
DATA_INTEGRITY	Processing of data integrity checks.

Each asynchronous task is automatically assigned an identifier which can be used to monitor the status of the task. This task identifier is returned by the API when you initiate an async task through the various async-enabled endpoints.

Monitoring a task

You can poll the task status through a GET request to the system tasks resource like this:

```
/api/33/system/tasks/{task-category-id}/{task-id}
```

An example request may look like this:

```
/api/33/system/tasks/DATAVALUE_IMPORT/j8Ki6TgreFw
```

The response will provide information about the status, such as the notification level, category, time and status. The *completed* property indicates whether the process is considered to be complete.

```
[
    {
        "uid": "hpiaeMy7wFX",
        "level": "INFO",
        "category": "DATAVALUE_IMPORT",
```

```
        "time": "2015-09-02T07:43:14.595+0000",
        "message": "Import done",
        "completed": true
    }
]
```

Monitoring all tasks for a category

You can poll all tasks for a specific category through a GET request to the system tasks resource:

```
/api/33/system/tasks/{task-category-id}
```

An example request to poll for the status of data value import tasks looks like this:

```
/api/33/system/tasks/DATAVALUE_IMPORT
```

Monitor all tasks

You can request a list of all currently running tasks in the system with a GET request to the system tasks resource:

```
/api/33/system/tasks
```

The response will look similar to this:

```
[
    {
        "EVENT_IMPORT": {},
        "DATA_STATISTICS": {},
        "RESOURCE_TABLE": {},
        "FILE_RESOURCE_CLEANUP": {},
        "METADATA_IMPORT": {},
        "CREDENTIALS_EXPIRY_ALERT": {},
        "SMS_SEND": {},
        "MOCK": {},
        "ANALYTICSTABLE_UPDATE": {},
        "COMPLETE_DATA_SET_REGISTRATION_IMPORT": {},
        "DATAVALUE_IMPORT": {},
        "DATA_SET_NOTIFICATION": {},
        "DATA_INTEGRITY": {
            "OB1qGRlCzap": [
                {
                    "uid": "LdHQK0PXZyF",
                    "level": "INFO",
                    "category": "DATA_INTEGRITY",
                    "time": "2018-03-26T15:02:32.171",
                    "message": "Data integrity checks completed in 38.31 seconds.",
                    "completed": true
                }
            ]
        },
        "PUSH_ANALYSIS": {},
        "MONITORING": {},
        "VALIDATION_RESULTS_NOTIFICATION": {},
        "REMOVE_EXPIRED_RESERVED_VALUES": {},
```

```
        "DATA_SYNC": {},
        "SEND_SCHEDULED_MESSAGE": {},
        "DATAVALUE_IMPORT_INTERNAL": {},
        "PROGRAM_NOTIFICATIONS": {},
        "META_DATA_SYNC": {},
        "ANALYTICS_TABLE": {},
        "PREDICTOR": {}
    }
]
```

View asynchronous task summaries

The task summaries resource allows you to retrieve a summary of an asynchronous task invocation. You need to specify the category and optionally the identifier of the task. The task identifier can be retrieved from the response of the API request which initiated the asynchronous task.

To retrieve the summary of a specific task you can issue a request to:

```
/api/33/system/taskSummaries/{task-category-id}/{task-id}
```

An example request might look like this:

```
/api/33/system/taskSummaries/DATAVALUE_IMPORT/k72jHfF13J1
```

The response will look similar to this:

```
{
    "responseType": "ImportSummary",
    "status": "SUCCESS",
    "importOptions": {
        "idSchemes": {},
        "dryRun": false,
        "async": true,
        "importStrategy": "CREATE_AND_UPDATE",
        "mergeMode": "REPLACE",
        "reportMode": "FULL",
        "skipExistingCheck": false,
        "sharing": false,
        "skipNotifications": false,
        "datasetAllowsPeriods": false,
        "strictPeriods": false,
        "strictCategoryOptionCombos": false,
        "strictAttributeOptionCombos": false,
        "strictOrganisationUnits": false,
        "requireCategoryOptionCombo": false,
        "requireAttributeOptionCombo": false,
        "skipPatternValidation": false
    },
    "description": "Import process completed successfully",
    "importCount": {
        "imported": 0,
        "updated": 431,
        "ignored": 0,
        "deleted": 0
    },
```

```
    "dataSetComplete": "false"
}
```

You might also retrieve import summaries for multiple tasks of a specific category with a request like this:

```
/api/33/system/taskSummaries/{task-category-id}
```

Get appearance information

You can retrieve the available flag icons in JSON format with a GET request:

```
/api/33/system/flags
```

You can retrieve the available UI styles in JSON format with a GET request:

```
/api/33/system/styles
```

Locales

DHIS2 supports translations both for the user interface and for database content.

UI locales

You can retrieve the available locales for the user interface through the following resource with a GET request. XML and JSON resource representations are supported.

```
/api/33/locales/ui
```

Database content locales

You can retrieve and create locales for the database content with GET and POST requests through the following resource. XML and JSON resource representations are supported.

```
/api/33/locales/db
```

Translations

DHIS2 allows for translations of database content. You can work with translations through the Web API using the *translations* resource.

```
/api/33/translations
```

Create a translation

You can create a translation with a POST request in JSON format:

```
{
    "objectId": "P3jJH5Tu5VC",
```

```
    "className": "DataElement",
    "locale": "es",
    "property": "name",
    "value": "Casos de fiebre amarilla"
}
```

```
POST /api/33/translations
```

The properties which support translations are listed in the table below.

Property names

Property name	Description
name	Object name
shortName	Object short name
description	Object description

The classes which support translations are listed in the table below.

Class names

Class name	Description
DataElementCategoryOption	Category option
DataElementCategory	Category
DataElementCategoryCombo	Category combination
DataElement	Data element
DataElementGroup	Data element group
DataElementGroupSet	Data element group set
Indicator	Indicator
IndicatorType	Indicator type
IndicatorGroup	Indicator group
IndicatorGroupSet	Indicator group set
OrganisationUnit	Organisation unit
OrganisationUnitGroup	Organisation unit group
OrganisationUnitGroupSet	Organisation unit group set
DataSet	Data set
Section	Data set section
ValidationRule	Validation rule
ValidationRuleGroup	Validation rule group
Program	Program
ProgramStage	Program stage
TrackedEntityAttribute	Tracked entity attribute
TrackedEntity	Tracked entity
RelationshipType	Relationship type for tracked entity instances
OptionSet	Option set

Class name	Description
Attribute	Attribute for metadata

Get translations

You can browse all translations through the translations resource:

```
GET /api/33/translations
```

You can use the standard filtering technique to fetch translations of interest. E.g. to get all translations for data elements in the Spanish locale you can use this request:

```
/api/33/translations.json?fields=*&filter=className:eq:DataElement&filter=locale:eq:es
```

To get translations for a specific object for all properties:

```
/api/33/translations.json?fields=*&filter=className:eq:DataElement
  &filter=locale:eq:fr&filter=objectId:eq:fbfJHSPpUQD
```

Short Message Service (SMS)

This section covers the SMS Web API for sending and receiving short text messages.

Outbound SMS service

The Web API supports sending outgoing SMS using the POST method. SMS can be sent to single or multiple destinations. One or more gateways need to be configured before using the service. An SMS will not be sent if there is no gateway configured. It needs a set of recipients and message text in JSON format as shown below.

```
/api/sms/outbound
```

```
{
    "message": "Sms Text",
    "recipients": ["004712341234", "004712341235"]
}
```

> **Note**
>
> Recipients list will be partitioned if the size exceeds `MAX_ALLOWED_RECIPIENTS` limit of 200.

The Web API also supports a query parameter version, but the parameterized API can only be used for sending SMS to a single destination.

```
/api/sms/outbound?message=text&recipient=004712341234
```

Outbound messages can be fetched using GET resource.

```
GET /api/sms/outbound
GET /api/sms/outbound?filter=status:eq:SENT
GET /api/sms/outbound?filter=status:eq:SENT&fields=*
```

Outbound messages can be deleted using DELETE resource.

```
DELETE /api/sms/outbound/{uid}
DELETE /api/sms/outbound?ids=uid1,uid2
```

Gateway response codes

Gateway may response with following response codes.

Gateway response codes

Response code	Response Message	Detail Description
RESULT_CODE_0	success	Message has been sent successfully
RESULT_CODE_1	scheduled	Message has been scheduled successfully
RESULT_CODE_22	internal fatal error	Internal fatal error
RESULT_CODE_23	authentication failure	Authentication credentials are incorrect
RESULT_CODE_24	data validation failed	Parameters provided in request are incorrect
RESULT_CODE_25	insufficient credits	Credit is not enough to send message
RESULT_CODE_26	upstream credits not available	Upstream credits not available
RESULT_CODE_27	exceeded your daily quota	You have exceeded your daily quota
RESULT_CODE_40	temporarily unavailable	Service is temporarily down
RESULT_CODE_201	maximum batch size exceeded	Maximum batch size exceeded
RESULT_CODE_200	success	The request was successfully completed

Response code	Response Message	Detail Description
RESULT_CODE_202	accepted	The message(s) will be processed
RESULT_CODE_207	multi-status	More than one message was submitted to the API; however, not all messages have the same status
RESULT_CODE_400	bad request	Validation failure (such as missing/invalid parameters or headers)
RESULT_CODE_401	unauthorized	Authentication failure. This can also be caused by IP lockdown settings
RESULT_CODE_402	payment required	Not enough credit to send message
RESULT_CODE_404	not found	Resource does not exist
RESULT_CODE_405	method not allowed	Http method is not support on the resource
RESULT_CODE_410	gone	Mobile number is blocked
RESULT_CODE_429	too many requests	Generic rate limiting error
RESULT_CODE_503	service unavailable	A temporary error has occurred on our platform - please retry

Inbound SMS service

The Web API supports collecting incoming SMS messages using the POST method. Incoming messages routed towards the DHIS2 Web API can be received using this API. The API collects inbound SMS messages and provides it to listeners for parsing, based on the SMS content (SMS Command). An example payload in JSON format is given below. Text, originator, received date and sent date are mandatory parameters. The rest are optional but the system will use the default value for these parameters.

```
/api/sms/inbound
```

```
{
    "text": "sample text",
    "originator": "004712341234",
    "gatewayid": "unknown",
    "receiveddate": "2016-05-01",
```

```
    "sentdate": "2016-05-01",
    "smsencoding": "1",
    "smsstatus": "1"
}
```

Inbound messages can be fetched using GET resourcef

```
GET /api/sms/inbound
GET /api/sms/inbound?fields=*&filter=smsstatus=INCOMING
```

Inbound messages can be deleted using DELETE resource

```
DELETE /api/sms/inbound/{uid}
DELETE /api/sms/inbound?ids=uid1,uid2
```

To import all un parsed messages POST /api/sms/inbound/import

User query parameters

Parameter	Type	Description
message	String	This is mandatory parameter which carries the actual text message.
originator	String	This is mandatory parameter which shows by whom this message was actually sent from.
gateway	String	This is an optional parameter which gives gateway id. If not present default text "UNKNOWN" will be stored
receiveTime	Date	This is an optional parameter. It is timestamp at which message was received at the gateway.

Gateway service administration

The Web API exposes resources which provide a way to configure and update SMS gateway configurations.

The list of different gateways configured can be retrieved using a GET method.

```
GET /api/33/gateways
```

Configurations can also be retrieved for a specific gateway type using GET method.

```
GET /api/33/gateways/{uid}
```

New gateway configurations can be added using POST. POST api requires type request parameter and currently its value can have either one *http,bulksms,clickatell,smpp*. First added gateway will be set to default. Only one gateway is allowed to be default at one time. Default gateway can only be changed through its api. If default gateway is removed then the next one the list will automatically becomes default.

```
POST /api/33/gateways
```

Configuration can be updated with by providing uid and gateway configurations as mentioned below

```
PUT /api/33/gateways/{uids}
```

Configurations can be removed for specific gateway type using DELETE method.

```
DELETE /api/33/gateways/{uid}
```

Default gateway can be retrieved and updated.

```
GET /api/33/gateways/default
```

Default gateway can be set using the PUT method.

```
PUT /api/33/gateways/default/{uid}
```

Gateway configuration

The Web API lets you create and update gateway configurations. For each type of gateway there are different parameters in the JSON payload. Sample JSON payloads for each gateway are given below. POST is used to create and PUT to update configurations. Header parameter can be used in case of GenericHttpGateway to send one or more parameter as http header.

Clickatell

```
{
    "type": "clickatell",
    "name": "clickatell",
    "username": "clickatelluser",
    "authToken": "XXXXXXXXXXXXXXXXXXXX",
    "urlTemplate": "https://platform.clickatell.com/messages"
}
```

Bulksms

```
{
    "type": "bulksms",
    "name": "bulkSMS",
    "username": "bulkuser",
    "password": "abc123"
}
```

SMPP Gateway

```
{
    "type": "smpp",
    "name": "smpp gateway2",
    "systemId": "smppclient1",
    "host": "localhost",
    "systemType": "cp",
    "numberPlanIndicator": "UNKNOWN",
    "typeOfNumber": "UNKNOWN",
    "bindType": "BIND_TX",
    "port": 2775,
    "password": "password",
    "compressed": false
}
```

Generic HTTP

```
{
    "type": "http",
    "name": "Generic",
    "configurationTemplate": "username=${username}&password=${password}&to=${recipients}
&countrycode=880&message=${text$}&messageid=0",
    "useGet": false,
    "sendUrlParameters": false,
    "contentType": "APPLICATION_JSON",
    "urlTemplate": "https://samplegateway.com/messages",
    "parameters": [
        {
            "header": true,
            "encode": false,
            "key": "username",
            "value": "user_uio",
            "confidential": true
        },
        {
            "header": true,
            "encode": false,
            "key": "password",
            "value": "123abcxyz",
            "confidential": true
        },
        {
            "header": false,
            "encode": false,
            "key": "deliveryReport",
            "value": "yes",
            "confidential": false
        }
    ],
    "isDefault": false
}
```

In generic http gateway any number of parameters can be added.

Generic SMS gateway parameters

Parameter	Type	Description
name	String	name of the gateway
configurationTemplate	String	Configuration template which get populated with parameter values. For example configuration template given above will be populated like this { "to": "+27001234567", "body": "Hello World!"}
useGet	Boolean	Http POST nethod will be used by default. In order to change it and Http GET, user can set useGet flag to true.
contentType	String	Content type specify what type of data is being sent. Supported types are APPLICATION_JSON, APPLICATION_XML, FORM_URL_ENCODED, TEXT_PLAIN
urlTemplate	String	Url template
header	Boolean	If parameter needs to be sent in Http headers
encode	Boolean	If parameter needs to be encoded
key	String	parameter key
value	String	parameter value
confidential	Boolean	If parameter is confidential. This parameter will not be exposed through API
sendUrlParameters	Boolean	If this flag is checked then urlTemplate can be appended with query parameters. This is usefull if gateway API only support HTTP GET. Sample urlTemplate looks like this "urlTemplate":"https://samplegateway.com/messages?apiKey={apiKey} &to={recipients},content={text},deliveryreport={dp}"

HTTP.OK will be returned if configurations are saved successfully otherwise *Error*

SMS Commands

SMS commands are being used to collect data through SMS. These commands belong to specific parser type. Each parser has different functionality.

The list of commands can be retrieved using GET.

```
GET /api/smsCommands
```

One particular command can be retrieved using GET.

```
GET /api/smsCommands/uid
```

One particular command can be updated using PUT.

```
PUT /api/smsCommands/uid
```

Command can be created using POST.

```
POST /api/smsCommands
```

One particular command can be deleted using DELETE.

```
DELETE /api/smsCommands/uid
```

SMS command types

Type	Usage
KEY_VALUE_PARSER	For aggregate data collection.
ALERT_PARSER	To send alert messages.
UNREGISTERED_PARSER	For disease surveillance case reporting.
TRACKED_ENTITY_REGISTRATION_PARSER	For tracker entity registration.
PROGRAM_STAGE_DATAENTRY_PARSER	Data collection for program stage. (TEI is identified based on phoneNumner)
EVENT_REGISTRATION_PARSER	Registration of single event. This is used for event programs.

SMS command types for Android

These command types can be used by the Android app for data submission via SMS when internet is unavailable. The SMS is composed by the Android app.

Type	Usage
AGGREGATE_DATASET	For aggregate data collection.
ENROLLMENT	For tracker entity registration.
TRACKER_EVENT	Event registration for tracker programs.
SIMPLE_EVENT	Event registration for event programs.
RELATIONSHIP	To create relationships.
DELETE	To delete event.

Program Messages

Program message lets you send messages to tracked entity instances, contact addresses associated with organisation units, phone numbers and email addresses. You can send messages through the `messages` resource.

```
/api/33/messages
```

Sending program messages

Program messages can be sent using two delivery channels:

- SMS (SMS)
- Email address (EMAIL)

Program messages can be sent to various recipients:

- Tracked entity instance: The system will look up attributes of value type PHONE_NUMBER or EMAIL (depending on the specified delivery channels) and use the corresponding attribute values.
- Organisation unit: The system will use the phone number or email information registered for the organisation unit.
- List of phone numbers: The system will use the explicitly defined phone numbers.
- List of email addresses: The system will use the explicitly defined email addresses.

Below is a sample JSON payload for sending messages using POST requests. Note that message resource accepts a wrapper object named `programMessages` which can contain any number of program messages.

```
POST /api/33/messages
```

```
{
    "programMessages": [
        {
            "recipients": {
                "trackedEntityInstance": {
                    "id": "UN810PwyVYO"
                },
                "organisationUnit": {
                    "id": "Rp268JB6Ne4"
                },
                "phoneNumbers": ["55512345", "55545678"],
                "emailAddresses": ["johndoe@mail.com", "markdoe@mail.com"]
            },
            "programInstance": {
                "id": "f3rg8gFag8j"
            },
            "programStageInstance": {
                "id": "pSllsjpfLH2"
            },
            "deliveryChannels": ["SMS", "EMAIL"],
            "subject": "Outbreak alert",
            "text": "An outbreak has been detected",
            "storeCopy": false
        }
    ]
}
```

The fields are explained in the following table.

Program message payload

Field	Required	Description	Values
recipients	Yes	Recipients of the program message. At least one recipient must be specified. Any number of recipients / types can be specified for a message.	Can be trackedEntityInstance, organisationUnit, an array of phoneNumbers or an array of emailAddresses.

Field	Required	Description	Values
programInstance	Either this or programStageInstance required	The program instance / enrollment.	Enrollment ID.
programStageInstance	Either this or programInstance required	The program stage instance / event.	Event ID.
deliveryChannels	Yes	Array of delivery channels.	SMS | EMAIL
subject	No	The message subject. Not applicable for SMS delivery channel.	Text.
text	Yes	The message text.	Text.
storeCopy	No	Whether to store a copy of the program message in DHIS2.	false (default) | true

A minimalistic example for sending a message over SMS to a tracked entity instance looks like this:

```
curl -d @message.json "https://play.dhis2.org/demo/api/33/messages"
  -H "Content-Type:application/json" -u admin:district
```

```
{
    "programMessages": [
        {
            "recipients": {
                "trackedEntityInstance": {
                    "id": "PQfMcpmXeFE"
                }
            },
            "programInstance": {
                "id": "JMgRZyeLWOo"
            },
            "deliveryChannels": ["SMS"],
            "text": "Please make a visit on Thursday"
        }
    ]
}
```

Retrieving and deleting program messages

The list of messages can be retrieved using GET.

```
GET /api/33/messages
```

To get list of all scheduled message

```
GET /api/33/messages/scheduled
GET /api/33/messages/scheduled?scheduledAt=2020-12-12
```

One particular message can also be retrieved using GET.

```
GET /api/33/messages/{uid}
```

Message can be deleted using DELETE.

```
DELETE /api/33/messages/{uid}
```

Querying program messages

The program message API supports program message queries based on request parameters. Messages can be filtered based on below mentioned query parameters. All requests should use the GET HTTP verb for retrieving information.

Query program messages API

Parameter	URL
programInstance	/api/33/messages?programInstance=6yWDMa0LP7
programStageInstance	/api/33/messages?programStageInstance=SllsjpfLH2
trackedEntityInstance	/api/33/messages?trackedEntityInstance=xdfejpfLH2
organisationUnit	/api/33/messages?ou=Sllsjdhoe3
processedDate	/api/33/messages?processedDate=2016-02-01

Users

This section covers the user resource methods.

```
/api/33/users
```

User query

The *users* resource offers additional query parameters beyond the standard parameters (e.g. paging). To query for users at the users resource you can use the following parameters.

User query parameters

Parameter	Type	Description
query	Text	Query value for first name, surname, username and email, case in-sensitive.
phoneNumber	Text	Query for phone number.
canManage	false | true	Filter on whether the current user can manage the returned users through the managed user group relationships.
authSubset	false | true	Filter on whether the returned users have a subset of the authorities of the current user.
lastLogin	Date	Filter on users who have logged in later than the given date.
inactiveMonths	Number	Filter on users who have not logged in for the given number of months.
inactiveSince	Date	Filter on users who have not logged in later than the given date.
selfRegistered	false | true	Filter on users who have self-registered their user account.

Parameter	Type	Description
invitationStatus	none | all | expired	Filter on user invitations, including all or expired invitations.
ou	Identifier	Filter on users who are associated with the organisation unit with the given identifier.
userOrgUnits	false | true	Filter on users who are associated with the organisation units linked to the currently logged in user.
includeChildren	false | true	Includes users from all children organisation units of the ou parameter.
page	Number	The page number.
pageSize	Number	The page size.

A query for max 10 users with "konan" as first name or surname (case in-sensitive) who have a subset of authorities compared to the current user:

```
/api/33/users?query=konan&authSubset=true&pageSize=10
```

User lookup

The user lookup API provides an endpoint for retrieving users where the response contains a minimal set of information. It does not require a specific authority and is suitable for allowing clients to look up information such as user first and surname, without exposing potentially sensitive user information.

```
/api/userLookup
```

The user lookup endpoint has two methods.

User lookup by identifier

You can do a user lookup by identifier using the following API request.

```
GET /api/userLookup/{id}
```

The user `id` will be matched against the following user properties in the specified order:

- UID
- UUID
- username

An example request looks like this:

```
/api/userLookup/QqvaU7JjkUV
```

The response will contain minimal information about a user.

```
{
    "id": "QqvaU7JjkUV",
```

```
    "username": "nkono",
    "firstName": "Thomas",
    "surname": "Nkono",
    "displayName": "Thomas Nkono"
}
```

User lookup query

You can make a query for users using the following API request.

```
GET /api/userLookup?query={string}
```

The `query` request parameter is mandatory. The query `string` will be matched against the following user properties:

- First name
- Surname
- Email
- Username

An example request looks like this:

```
/api/userLookup?query=John
```

The response will contain information about the users matching the request.

```
{
    "users": [
        {
            "id": "DXyJmlo9rge",
            "username": "jbarnes",
            "firstName": "John",
            "surname": "Barnes",
            "displayName": "John Barnes"
        },
        {
            "id": "N3PZBUlN8vq",
            "username": "jkamara",
            "firstName": "John",
            "surname": "Kamara",
            "displayName": "John Kamara"
        }
    ]
}
```

User account create and update

Creating and updating users are supported through the API. A basic payload to create a user looks like the below example. Note that the password will be sent in plain text so remember to enable SSL/HTTPS for network transport.

```
{
    "id": "Mj8balLULKp",
    "firstName": "John",
```

```
    "surname": "Doe",
    "email": "johndoe@mail.com",
    "userCredentials": {
        "id": "lWCkJ4etppc",
        "userInfo": {
            "id": "Mj8balLULKp"
        },
        "username": "johndoe123",
        "password": "Your-password-123",
        "skype": "john.doe",
        "telegram": "joh.doe",
        "whatsApp": "+1-541-754-3010",
        "facebookMessenger": "john.doe",
        "avatar": {
            "id": "<fileResource id>"
        },
        "userRoles": [
            {
                "id": "Ufph3mGRmMo"
            }
        ]
    },
    "organisationUnits": [
        {
            "id": "Rp268JB6Ne4"
        }
    ],
    "userGroups": [
        {
            "id": "wl5cDMuUhmF"
        }
    ]
}
```

```
curl -X POST -d @u.json "http://server/api/33/users" -u user:pass
  -H "Content-Type: application/json"
```

In the user creation payload, user groups are only supported when importing or *POSTing* a single user at a time. If you attempt to create more than one user while specifiying user groups, you will not recieve an error and the users will be created but no user groups will be assigned. This is by design and is limited because of the many-to-many relationship between users and user groups whereby user groups is the owner of the relationship. To update or create mulitple users and their user groups, consider a program to *POST* one at a time, or *POST* all users followed by another action to update their user groups while specifiying the new user's identifiers.

After the user is created, a *Location* header is sent back with the newly generated ID (you can also provide your own using the `/api/system/id` endpoint). The same payload can then be used to do updates, but remember to then use *PUT* instead of *POST* and the endpoint is now `/api/users/ID`.

```
curl -X PUT -d @u.json "http://server/api/33/users/ID" -u user:pass
  -H "Content-Type: application/json"
```

For more info about the full payload available, please see `/api/schemas/user`.

For more info about uploading and retrieving user avatars, please see the `/fileResources` endpoint.

User account invitations

The Web API supports inviting people to create user accounts through the `invite` resource. To create an invitation you should POST a user in XML or JSON format to the invite resource. A specific username can be forced by defining the username in the posted entity. By omitting the username, the person will be able to specify it herself. The system will send out an invitation through email. This requires that email settings have been properly configured.

The invite resource is useful in order to securely allow people to create accounts without anyone else knowing the password or by transferring the password in plain text. The payload to use for the invite is the same as for creating users. An example payload in JSON looks like this:

```
{
    "firstName": "John",
    "surname": "Doe",
    "email": "johndoe@mail.com",
    "userCredentials": {
        "username": "johndoe",
        "userRoles": [
            {
                "id": "Euq3XfEIEbx"
            }
        ]
    },
    "organisationUnits": [
        {
            "id": "ImspTQPwCqd"
        }
    ],
    "userGroups": [
        {
            "id": "vAvEltyXGbD"
        }
    ]
}
```

The user invite entity can be posted like this:

```
curl -d @invite.json "localhost/api/33/users/invite" -u admin:district
  -H "Content-Type:application/json"
```

To send out invites for multiple users at the same time you must use a slightly different format. For JSON:

```
{
    "users": [
        {
            "firstName": "John",
            "surname": "Doe",
            "email": "johndoe@mail.com",
            "userCredentials": {
                "username": "johndoe",
                "userRoles": [
                    {
                        "id": "Euq3XfEIEbx"
                    }
                ]
```

```
            },
            "organisationUnits": [
                {
                    "id": "ImspTQPwCqd"
                }
            ]
        },
        {
            "firstName": "Tom",
            "surname": "Johnson",
            "email": "tomj@mail.com",
            "userCredentials": {
                "userRoles": [
                    {
                        "id": "Euq3XfEIEbx"
                    }
                ]
            },
            "organisationUnits": [
                {
                    "id": "ImspTQPwCqd"
                }
            ]
        }
    ]
}
```

To create multiple invites you can post the payload to the api/users/invites resource like this:

```
curl -d @invites.json "localhost/api/33/users/invites" -u admin:district
  -H "Content-Type:application/json"
```

There are certain requirements for user account invitations to be sent out:

- Email SMTP server must be configured properly on the server.
- The user to be invited must have specified a valid email.
- If username is specified it must not be already taken by another existing user.

If any of these requirements are not met the invite resource will return with a *409 Conflict* status code together with a descriptive message.

User replication

To replicate a user you can use the *replica* resource. Replicating a user can be useful when debugging or reproducing issues reported by a particular user. You need to provide a new username and password for the replicated user which you will use to authenticate later. Note that you need the ALL authority to perform this action. To replicate a user you can post a JSON payload looking like below:

```
{
    "username": "user_replica",
    "password": "SecretPassword"
}
```

This payload can be posted to the replica resource, where you provide the identifier of the user to replicate in the URL:

```
/api/33/users/<uid>/replica
```

An example of replicating a user using curl looks like this:

```
curl -d @replica.json "localhost/api/33/users/N3PZBUlN8vq/replica"
  -H "Content-Type:application/json" -u admin:district
```

Current user information

In order to get information about the currently authenticated user and its associations to other resources you can work with the *me* resource (you can also refer to it by its old name *currentUser*). The current user related resources gives your information which is useful when building clients for instance for data entry and user management. The following describes these resources and their purpose.

Provides basic information about the user that you are currently logged in as, including username, user credentials, assigned organisation units:

```
/api/me
```

Gives information about currently unread messages and interpretations:

```
/api/me/dashboard
```

In order to change password, this end point can be used to validate newly entered password. Password validation will be done based on PasswordValidationRules configured in the system. This end point support POST and password string should be sent in POST body.

```
/api/me/validatePassword
```

While changing password, this end point (support POST) can be used to verify old password. Password string should be sent in POST body.

```
/api/me/verifyPassword
```

Returns the set of authorities granted to the current user:

```
/api/me/authorization
```

Returns true or false, indicating whether the current user has been granted the given `<auth>` authorization:

```
/api/me/authorization/<auth>
```

Gives the data approval levels which are relevant to the current user:

```
/api/me/dataApprovalLevels
```

System settings

You can manipulate system settings by interacting with the *systemSettings* resource. A system setting is a simple key-value pair, where both the key and the value are plain text strings. To save or update a system setting you can make a *POST* request to the following URL:

```
/api/33/systemSettings/my-key?value=my-val
```

Alternatively, you can submit the setting value as the request body, where content type is set to "text/plain". As an example, you can use curl like this:

```
curl "play.dhis2.org/demo/api/33/systemSettings/my-key" -d "My long value"
  -H "Content-Type: text/plain" -u admin:district
```

To set system settings in bulk you can send a JSON object with a property and value for each system setting key-value pair using a POST request:

```
{
    "keyApplicationNotification": "Welcome",
    "keyApplicationIntro": "DHIS2",
    "keyApplicationFooter": "Read more at dhis2.org"
}
```

Translations for translatable Setting keys can be set by specifying locale as a query parameter and translated value which can be specified either as a query param or withing the body payload. See an example URL:

```
/api/33/systemSettings/<my-key>?locale=<my-locale>&value=<my-translated-value>
```

You should replace my-key with your real key and my-val with your real value. To retrieve the value for a given key (in JSON or plain text) you can make a *GET* request to the following URL:

```
/api/33/systemSettings/my-key
```

Alternatively, you can specify the key as a query parameter:

```
/api/33/systemSettings?key=my-key
```

You can retrieve specific system settings as JSON by repeating the key query parameter:

```
curl "play.dhis2.org/demo/api/33/systemSettings?
key=keyApplicationNotification&key=keyApplicationIntro"
  -u admin:district
```

You can retrieve all system settings with a GET request:

```
/api/33/systemSettings
```

To retrieve a specific translation for a given translatable key you can specify a locale as query param:

```
/api/33/systemSettings/<my-key>?locale=<my-locale>
```

If present, the translation for the given locale is returned. Otherwise, a default value is returned. If no locale is specified for the translatable key, the user default UI locale is used to fetch the correct translation. If the given translation is not present, again, the default value is returned.

The priority for translatable keys is the following:

```
specified locale > user's default UI locale > defaut value
```

To delete a system setting, you can make a *DELETE* request to the URL similar to the one used above for retrieval. If a translatable key is used, all present translations will be deleted as well.

To delete only a specific translation of translatable key, the same URL as for adding a translation should be used and the empty value should be provided:

```
/api/33/systemSettings/<my-key>?locale=<my-locale>&value=
```

The available system settings are listed below.

System settings

Key	Description	Translatable
keyUiLocale	Locale for the user interface	No
keyDbLocale	Locale for the database	No
keyAnalysisDisplayProperty	The property to display in analysis. Default: "name"	No
keyAnalysisDigitGroupSeparator	The separator used to separate digit groups	No
keyCurrentDomainType	Not yet in use	No
keyTrackerDashboardLayout	Used by tracker capture	No
applicationTitle	The application title. Default: "DHIS2"	Yes
keyApplicationIntro	The application introduction	Yes
keyApplicationNotification	Application notification	Yes
keyApplicationFooter	Application left footer	Yes
keyApplicationRightFooter	Application right footer	Yes
keyFlag	Application flag	No
keyFlagImage	Flag used in dashboard menu	No

Key	Description	Translatable
startModule	The startpage of the application. Default: "dhis-web-dashboard-integration"	No
factorDeviation	Data analysis standard deviation factor. Default: "2d"	No
keyEmailHostName	Email server hostname	No
keyEmailPort	Email server port	No
keyEmailTls	Use TLS. Default: "true"	No
keyEmailSender	Email sender	No
keyEmailUsername	Email server username	No
keyEmailPassword	Email server password	No
minPasswordLength	Minimum length of password	No
maxPasswordLength	Maximum length of password	No
keySmsSetting	SMS configuration	No
keyCacheStrategy	Cache strategy. Default: "CACHE_6AM_TOMORROW"	No
keyCacheability	PUBLIC or PRIVATE. Determines if proxy servers are allowed to cache data or not.	No
phoneNumberAreaCode	Phonenumber area code	No
multiOrganisationUnitForms	Enable multi-organisation unit forms. Default: "false"	No
keyConfig		No
keyAccountRecovery	Enable user account recovery. Default: "false"	No
keyLockMultipleFailedLogins	Enable locking access after multiple failed logins	No
googleAnalyticsUA	Google Analytic UA key for tracking site-usage	No
credentialsExpires	Require user account password change. Default: "0" (Never)	No
credentialsExpiryAlert	Enable alert when credentials are close to expiration date	No
keySelfRegistrationNoRecaptcha	Do not require recaptcha for self registration. Default: "false"	No
recaptchaSecret	Google API recaptcha secret. Default: dhis2 play instance API secret, but this will only works on you local instance and not in production.	No
recaptchaSite	Google API recaptcha site. Default: dhis2 play instance API site, but this will only works on you local instance and not in production.	No
keyCanGrantOwnUserAuthorityGroups	Allow users to grant own user roles. Default: "false"	No
keySqlViewMaxLimit	Max limit for SQL view	No

Key	Description	Translatable
keyRespectMetaDataStartEndDatesInAnalyticsTableExport	When "true", analytics will skip data not within category option's start and end dates. Default: "false"	No
keySkipDataTypeValidationInAnalyticsTableExport	Skips data type validation in analytics table export	No
keyCustomLoginPageLogo	Logo for custom login page	No
keyCustomTopMenuLogo	Logo for custom top menu	No
keyCacheAnalyticsDataYearThreshold	Analytics data older than this value (in years) will always be cached. "0" disabled this setting. Default: 0	No
keyCacheAnalyticsDataYearThreshold	Analytics data older than this value (in years) will always be cached. "0" disabled this setting. Default: 0	No
analyticsFinancialYearStart	Set financial year start. Default: October	No
keyIgnoreAnalyticsApprovalYearThreshold	"0" check approval for all data. "-1" disable approval checking. "1" or higher checks approval for all data that is newer than "1" year.	No
keyAnalyticsMaxLimit	Maximum number of analytics recors. Default: "50000"	No
keyAnalyticsMaintenanceMode	Put analytics in maintenance mode. Default: "false"	No
keyDatabaseServerCpus	Number of database server CPUs. Default: "0" (Automatic)	No
keyLastSuccessfulAnalyticsTablesRuntime	Keeps timestamp of last successful analytics tables run	No
keyLastSuccessfulLatestAnalyticsPartitionRuntime	Keeps timestamp of last successful latest analytics partition run	No
keyLastMonitoringRun	Keeps timestamp of last monitoring run	No
keyLastSuccessfulDataSynch	Keeps timestamp of last successful data values synchronization	No
keyLastSuccessfulEventsDataSynch	Keeps timestamp of last successful Event programs data synchronization	No
keyLastCompleteDataSetRegistrationSyncSuccess	Keeps timestamp of last successful completeness synchronization	No
syncSkipSyncForDataChangedBefore	Specifies timestamp used to skip synchronization of all the data changed before this point in time	No
keyLastSuccessfulAnalyticsTablesUpdate	Keeps timestamp of last successful analytics tables update	No
keyLastSuccessfulLatestAnalyticsPartitionUpdate	Keeps timestamp of last successful latest analytics partition update	No
keyLastSuccessfulResourceTablesUpdate	Keeps timestamp of last successful resource tables update	No

Key	Description	Translatable
keyLastSuccessfulSystemMonitoringPush	Keeps timestamp of last successful system monitoring push	No
keyLastSuccessfulMonitoring	Keeps timestamp of last successful monitoring	No
keyNextAnalyticsTableUpdate	Keeps timestamp of next analytics table update	No
helpPageLink	Link to help page. Default: "https://dhis2.github.io/dhis2-docs/master/en/user/html/dhis2_user_manual_en.html	No
keyAcceptanceRequiredForApproval	Acceptance required before approval. Default: "false"	No
keySystemNotificationsEmail	Where to email system notifications	No
keyAnalysisRelativePeriod	Default relative period for analysis. Default: "LAST_12_MONTHS"	No
keyRequireAddToView	Require authority to add to view object lists. Default: "false"	No
keyAllowObjectAssignment	Allow assigning object to related objects during add or update. Default: "false"	No
keyUseCustomLogoFront	Enables the usage of a custom logo on the front page. Default: "false"	No
keyUseCustomLogoBanner	Enables the usage of a custom banner on the website. Default: "false"	No
keyDataImportStrictPeriods		No
keyDataImportStrictPeriods	Require periods to match period type of data set. Default: "false"	No
keyDataImportStrictDataElements	Require data elements to be part of data set. Default: "false"	No
keyDataImportStrictCategoryOptionCombos	Require category option combos to match category combo of data element. Default: "false"	No
keyDataImportStrictOrganisationUnits	Require organisation units to match assignment of data set. Default: "false"	No
keyDataImportStrictAttributeOptionsCombos	Require attribute option combis to match category combo of data set. Default: "false"	No
keyDataImportRequireCategoryOptionCombo	Require category option combo to be specified. Default: "false"	No
keyDataImportRequireAttributeOptionCombo	Require attribute option combo to be specified. Default: "false"	No
keyCustomJs	Custom JavaScript to be used on the website	No
keyCustomCss	Custom CSS to be used on the website	No
keyCalendar	The calendar type. Default: "iso8601".	No
keyDateFormat	The format in which dates should be displayed. Default: "yyyy-MM-dd".	No

Key	Description	Translatable
keyStyle	The style used on the DHIS2 webpages. Default: "light_blue/light_blue.css".	No
keyRemoteInstanceUrl	Url used to connect to remote instance	No
keyRemoteInstanceUsername	Username used to connect to remote DHIS2 instance	No
keyRemoteInstancePassword	Password used to connect to remote DHIS2 instance	No
keyGoogleMapsApiKey	Google Maps API key	No
keyGoogleCloudApiKey	Google Cloud API key	No
keyLastMetaDataSyncSuccess	Keeps timestamp of last successful metadata synchronization	No
keyVersionEnabled	Enables metadata versioning	No
keyMetadataFailedVersion	Keeps details about failed metadata version sync	No
keyMetadataLastFailedTime	Keeps timestamp of last metadata synchronization failure	No
keyLastSuccessfulScheduledProgramNotifications	Not in use	No
keyLastSuccessfulScheduledDataSetNotifications	Not in use	No
keyRemoteMetadataVersion	Details about metadata version of remote instance	No
keySystemMetadataVersion	Details about metadata version of the system	No
keyStopMetadataSync	Flag to stop metadata synchronization	No
keyFileResourceRetentionStrategy	Determines how long file resources associated with deleted or updated values are kept. NONE, THREE_MONTHS, ONE_YEAR, or FOREVER.	No
syncMaxRemoteServerAvailabilityCheckAttempts	Specifies how many times the availability of remote server will be checked before synchronization jobs fail.	No
syncMaxAttempts	Specifies max attempts for synchronization jobs	No
syncDelayBetweenRemoteServerAvailabilityCheckAttempts	Delay between remote server availability checks	No
lastSuccessfulDataStatistics	Keeps timestamp of last successful data analytics	No
keyHideDailyPeriods	Not in use	No
keyHideWeeklyPeriods	Not in use	No
keyHideMonthlyPeriods	Not in use	No
keyHideBiMonthlyPeriods	Not in use	No

User settings

You can manipulate user settings by interacting with the *userSettings* resource. A user setting is a simple key-value pair, where both the key and the value are plain text strings. The user setting will be linked to the user who is authenticated for the Web API request. To return a list of all user settings, you can send a *GET* request to the following URL:

```
/api/33/userSettings
```

User settings not set by the user, will fall back to the equivalent system setting. To only return the values set explicitly by the user, you can append ?useFallback=false to the above URL, like this:

```
/api/33/userSettings?useFallback=false
```

To save or update a setting for the currently authenticated user you can make a *POST* request to the following URL:

```
/api/33/userSettings/my-key?value=my-val
```

You can specify the user for which to save the setting explicitly with this syntax:

```
/api/33/userSettings/my-key?user=username&value=my-val
```

Alternatively, you can submit the setting value as the request body, where content type is set to "text/plain". As an example, you can use curl like this:

```
curl "https://play.dhis2.org/demo/api/33/userSettings/my-key" -d "My long value"
  -H "Content-Type: text/plain" -u admin:district
```

As an example, to set the UI locale of the current user to French you can use the following command.

```
curl "https://play.dhis2.org/demo/api/33/userSettings/keyUiLocale?value=fr"
  -X POST -u admin:district
```

You should replace my-key with your real key and my-val with your real value. To retrieve the value for a given key in plain text you can make a *GET* request to the following URL:

```
/api/33/userSettings/my-key
```

To delete a user setting, you can make a *DELETE* request to the URL similar to the one used above for retrieval.

The available system settings are listed below.

User settings

Key	Options	Description
keyStyle	light_blue/light_blue.css | green/green.css | vietnam/vietnam.css	User interface stylesheet.
keyMessageEmailNotification	false | true	Whether to send email notifications.
keyMessageSmsNotification	false | true	Whether to send SMS notifications.
keyUiLocale	Locale value	User interface locale.
keyDbLocale	Locale value	Database content locale.
keyAnalysisDisplayProperty	name | shortName	Property to display for metadata in analysis apps.
keyCurrentDomainType	all | aggregate | tracker	Data element domain type to display in lists.
keyAutoSaveCaseEntryForm	false | true	Save case entry forms periodically.
keyAutoSaveTrackedEntityForm	false | true	Save person registration forms periodically.
keyAutoSaveDataEntryForm	false | true	Save aggregate data entry forms periodically.
keyTrackerDashboardLayout	false | true	Tracker dashboard layout.

Organisation units

The *organisationUnits* resource follows the standard conventions as other metadata resources in DHIS2. This resource supports some additional query parameters.

Get list of organisation units

To get a list of organisation units you can use the following resource.

```
/api/33/organisationUnits
```

Organisation units query parameters

Query parameter	Options	Description
userOnly	false | true	Data capture organisation units associated with current user only.
userDataViewOnly	false | true	Data view organisation units associated with current user only.
userDataViewFallback	false | true	Data view organisation units associated with current user only with fallback to data capture organisation units.
query	string	Query against the name, code and ID properties.
level	integer	Organisation units at the given level in the hierarchy.
maxLevel	integer	Organisation units at the given max level or levels higher up in the hierarchy.

Query parameter	Options	Description
withinUserHierarchy	false | true	Limits search and retrieval to organisation units that are within the users data capture scope.
withinUserSearchHierarchy	false | true	Limits search and retrieval to organisation units that are within the current users search scope. Note: "withinUserHierarchy", if true, takes higher precedence.
memberCollection	string	For displaying count of members within a collection, refers to the name of the collection associated with organisation units.
memberObject	UID	For displaying count of members within a collection, refers to the identifier of the object member of the collection.

Get organisation unit with relations

To get an organisation unit with related organisation units you can use the following resource.

```
/api/33/organisationUnits/{id}
```

Organisation unit parameters

Query parameter	Options	Description
includeChildren	false | true	Include immediate children of the specified organisation unit, i.e. the units at the immediate level below in the subhierarchy.
includeDescendants	false | true	Include all children of the specified organisation unit, i.e. all units in the sub-hierarchy.
includeAncestors	false | true	Include all parents of the specified organisation unit.
level	integer	Include children of the specified organisation unit at the given level of the sub-hierarchy (relative to the organisation unit where the immediate level below is level 1).

Data sets

The *dataSets* resource follows the standard conventions as other metadata resources in DHIS2. This resource supports some additional query parameters.

```
/api/33/dataSets
```

To retrieve the version of a data set you can issue a GET request:

```
GET /api/33/dataSets/<uid>/version
```

To bump (increase by one) the version of a data set you can issue a POST request:

```
POST /api/33/dataSets/<uid>/version
```

DataSet Notification Template

The *dataset notification templates* resource follows the standard conventions as other metadata resources in DHIS2.

```
GET /api/33/dataSetNotficationTemplates
```

To retrieve data set notification template you can issue a GET request:

```
GET /api/33/dataSetNotficationTemplates/<uid>
```

To add data set notification template you can issue a POST request:

```
POST /api/33/dataSetNotficationTemplates
```

To delete data set notification template you can issue a DELETE request:

```
DELETE /api/33/dataSetNotficationTemplates/<uid>
```

JSON payload sample is given below:

```
{
    "name": "dataSetNotificationTemplate1",
    "notificationTrigger": "COMPLETION",
    "relativeScheduledDays": 0,
    "notificationRecipient": "ORGANISATION_UNIT_CONTACT",
    "dataSets": [
        {
            "id": "eZDhcZi6FLP"
        }
    ],
    "deliveryChannels": ["SMS"],
    "subjectTemplate": "V{data_name}",
    "messageTemplate": "V{data_name}V{complete_registration_period}",
    "sendStrategy": "SINGLE_NOTIFICATION"
}
```

Filled organisation unit levels

The *filledOrganisationUnitLevels* resource provides an ordered list of organisation unit levels, where generated levels are injected into the list to fill positions for which it does not exist a persisted level.

```
GET /api/33/filledOrganisationUnitLevels
```

To set the organisation unit levels you can issue a POST request with a JSON payload looking like this.

```
{
    "organisationUnitLevels": [
```

```
        {
            "name": "National",
            "level": 1,
            "offlineLevels": 3
        },
        {
            "name": "District",
            "level": 2
        },
        {
            "name": "Chiefdom",
            "level": 3
        },
        {
            "name": "Facility",
            "level": 4
        }
    ]
}
```

To do functional testing with curl you can issue the following command.

```
curl "http://localhost/api/33/filledOrganisationUnitLevels" -H "Content-Type:application/json"
  -d @levels.json -u admin:district
```

Static content

The *staticContent* resource allows you to upload and retrieve custom logos used in DHIS2. The resource lets the user upload a file with an associated key, which can later be retrieved using the key. Only PNG files are supported and can only be uploaded to the `logo_banner` and `logo_front` keys.

```
/api/33/staticContent
```

Static content keys

Key	Description
logo_banner	Logo in the application top menu on the left side.
logo_front	Logo on the login-page above the login form.

To upload a file, send the file with a *POST* request to:

```
POST /api/33/staticContent/<key>
```

Example request to upload logo.png to the `logo_front` key:

```
curl -F "file=@logo.png;type=image/png" "https://play.dhis2.org/demo/api/33/staticContent/
logo_front"
  -X POST -H "Content-Type: multipart/form-data" -u admin:district
```

Uploading multiple files with the same key will overwrite the existing file. This way, retrieving a file for any given key will only return the latest file uploaded.

To retrieve a logo, you can *GET* the following:

```
GET /api/33/staticContent/<key>
```

Example of requests to retrieve the file stored for `logo_front`:

- Adding "Accept: text/html" to the HTTP header.*__ In this case, the endpoint will return a default image if nothing is defined. Will return an image stream when a custom or default image is found.

```
curl "https://play.dhis2.org/demo/api/33/staticContent/logo_front"
  -H "Accept: text/html" -L -u admin:district
```

- Adding "Accept: application/json" to the HTTP header.*__ With this parameter set, the endpoint will never return a default image if the custom logo is not found. Instead, an error message will be returned. When the custom image is found this endpoint will return a JSON response containing the path/URL to the respective image.

```
curl "https://play.dhis2.org/demo/api/33/staticContent/logo_front"
  -H "Accept: application/json" -L -u admin:district
```

Success and error messages will look like this:

```
{
    "images": {
        "png": "http://localhost:8080/dhis/api/staticContent/logo_front"
    }
}
```

```
{
    "httpStatus": "Not Found",
    "httpStatusCode": 404,
    "status": "ERROR",
    "message": "No custom file found."
}
```

To use custom logos, you need to enable the corresponding system settings by setting it to *true*. If the corresponding setting is false, the default logo will be served.

Configuration

To access configuration you can interact with the *configuration* resource. You can get XML and JSON responses through the *Accept* header or by using the .json or .xml extensions. You can *GET* all properties of the configuration from:

```
/api/33/configuration
```

You can send *GET* and *POST* requests to the following specific resources:

```
GET /api/33/configuration/systemId

GET POST DELETE /api/33/configuration/feedbackRecipients

GET POST DELETE /api/33/configuration/offlineOrganisationUnitLevel

GET POST /api/33/configuration/infrastructuralDataElements

GET POST /api/33/configuration/infrastructuralIndicators

GET POST /api/33/configuration/infrastructuralPeriodType

GET POST DELETE /api/33/configuration/selfRegistrationRole

GET POST DELETE /api/33/configuration/selfRegistrationOrgUnit
```

For the CORS whitelist configuration you can make a POST request with an array of URLs to whitelist as payload using "application/json" as content-type, for instance:

```
["www.google.com", "www.dhis2.org", "www.who.int"]
```

```
GET POST /api/33/configuration/corsWhitelist
```

For POST requests, the configuration value should be sent as the request payload as text. The following table shows appropriate configuration values for each property.

Configuration values

Configuration property	Value
feedbackRecipients	User group ID
offlineOrganisationUnitLevel	Organisation unit level ID
infrastructuralDataElements	Data element group ID
infrastructuralIndicators	Indicator group ID
infrastructuralPeriodType	Period type name (e.g. "Monthly")
selfRegistrationRole	User role ID
selfRegistrationOrgUnit	Organisation unit ID
smtpPassword	SMTP email server password
remoteServerUrl	URL to remote server
remoteServerUsername	Username for remote server authentication
remoteServerPassword	Password for remote server authentication
corsWhitelist	JSON list of URLs

As an example, to set the feedback recipients user group you can invoke the following curl command:

```
curl "localhost/api/33/configuration/feedbackRecipients" -d "wl5cDMuUhmF"
  -H "Content-Type:text/plain"-u admin:district
```

Read-Only configuration service

To access configuration you can now use read-only service. This service will provide read-only access to *UserSettings, SystemSettings and DHIS2 server configurations* You can get XML and JSON responses through the *Accept* header. You can *GET* all settings from:

```
/api/33/configuration/settings
```

You can get filtered settings based on setting type:

```
GET /api/33/configuration/settings/filter?type=USER_SETTING

GET /api/33/configuration/settings/filter?type=CONFIGURATION
```

More than one type can be provided

```
GET /api/33/configuration/settings/filter?type=USER_SETTING&type=SYSTEM_SETTING
```

SettingType values

Value	Description
USER_SETTING	To get user settings
SYSTEM_SETTING	To get system settings
CONFIGURATION	To get DHIS server settings

> **Note**
>
> Fields which are confidential will be provided in the output but without values.

Internationalization

In order to retrieve key-value pairs for translated strings you can use the *i18n* resource.

```
/api/33/i18n
```

The endpoint is located at */api/i18n* and the request format is a simple array of the key-value pairs:

```
["access_denied", "uploading_data_notification"]
```

The request must be of type *POST* and use *application/json* as content-type. An example using curl, assuming the request data is saved as a file `keys.json`:

```
curl -d @keys.json "play.dhis2.org/demo/api/33/i18n" -X POST
  -H "Content-Type: application/json" -u admin:district
```

The result will look like this:

```
{
    "access_denied": "Access denied",
    "uploading_data_notification": "Uploading locally stored data to the server"
}
```

SVG conversion

The Web API provides a resource which can be used to convert SVG content into more widely used formats such as PNG and PDF. Ideally this conversion should happen on the client side, but not all client side technologies are capable of performing this task. Currently PNG and PDF output formats are supported. The SVG content itself should be passed with a *svg* query parameter, and an optional query parameter *filename* can be used to specify the filename of the response attachment file. Note that the file extension should be omitted. For PNG you can send a *POST* request to the following URL with Content-type `application/x-www-form-urlencoded`, identical to a regular HTML form submission.

```
api/svg.png
```

For PDF you can send a *POST* request to the following URL with content-type `application/x-www-form-urlencoded`.

```
api/svg.pdf
```

Query parameters

Query parameter	Required	Description
svg	Yes	The SVG content
filename	No	The file name for the returned attachment without file extension

Tracker Web API

Tracker Web API consists of 3 endpoints that have full CRUD (create, read, update, delete) support. The 3 endpoints are `/api/trackedEntityInstances`, `/api/enrollments` and `/api/events` and they are responsible for tracked entity instance, enrollment and event items.

Tracked entity instance management

Tracked entity instances have full CRUD support in the API. Together with the API for enrollment most operations needed for working with tracked entity instances and programs are supported.

```
/api/33/trackedEntityInstances
```

Creating a new tracked entity instance

For creating a new person in the system, you will be working with the *trackedEntityInstances* resource. A template payload can be seen below:

```
{
    "trackedEntity": "tracked-entity-id",
    "orgUnit": "org-unit-id",
    "geometry": "<Geo JSON>",
    "attributes": [
        {
            "attribute": "attribute-id",
            "value": "attribute-value"
        }
    ]
}
```

The field "geometry" accepts a GeoJson object, where the type of the GeoJson have to match the featureType of the TrackedEntityType definition. An example GeoJson object looks like this:

```
{
    "type": "Point",
    "coordinates": [1, 1]
}
```

The "coordinates" field was introduced in 2.29, and accepts a coordinate or a polygon as a value.

For getting the IDs for `relationship` and `attributes` you can have a look at the respective resources `relationshipTypes`, `trackedEntityAttributes`. To create a tracked entity instance you must use the HTTP *POST* method. You can post the payload the following URL:

```
/api/trackedEntityInstances
```

For example, let us create a new instance of a person tracked entity and specify its first name and last name attributes:

```
{
    "trackedEntity": "nEenWmSyUEp",
    "orgUnit": "DiszpKrYNg8",
    "attributes": [
        {
            "attribute": "w75KJ2mc4zz",
            "value": "Joe"
        },
        {
            "attribute": "zDhUuAYrxNC",
            "value": "Smith"
        }
    ]
}
```

To push this to the server you can use the cURL command like this:

```
curl -d @tei.json "https://play.dhis2.org/demo/api/trackedEntityInstances" -X POST
  -H "Content-Type: application/json" -u admin:district
```

To create multiple instances in one request you can wrap the payload in an outer array like this and POST to the same resource as above:

```
{
    "trackedEntityInstances": [
        {
            "trackedEntity": "nEenWmSyUEp",
            "orgUnit": "DiszpKrYNg8",
            "attributes": [
                {
                    "attribute": "w75KJ2mc4zz",
                    "value": "Joe"
                },
                {
                    "attribute": "zDhUuAYrxNC",
                    "value": "Smith"
                }
            ]
        },
        {
            "trackedEntity": "nEenWmSyUEp",
            "orgUnit": "DiszpKrYNg8",
            "attributes": [
                {
                    "attribute": "w75KJ2mc4zz",
                    "value": "Jennifer"
                },
                {
                    "attribute": "zDhUuAYrxNC",
                    "value": "Johnson"
                }
            ]
        }
    ]
}
```

The system does not allow the creation of a tracked entity instance (as well as enrollment and event) with a UID that was already used in the system. That means that UIDs cannot be reused.

Updating a tracked entity instance

For updating a tracked entity instance, the payload is equal to the previous section. The difference is that you must use the HTTP *PUT* method for the request when sending the payload. You will also need to append the person identifier to the *trackedEntityInstances* resource in the URL like this, where `<tracked-entity-instance-identifier>` should be replaced by the identifier of the tracked entity instance:

```
/api/trackedEntityInstances/<tracked-entity-instance-id>
```

The payload has to contain all, even non-modified, attributes and relationships. Attributes or relationships that were present before and are not present in the current payload any more will be removed from the system. This means that if attributes/relationships are empty in the current payload, all existing attributes/relationships will be deleted from the system. From 2.31, it is possible to ignore empty attributes/relationships in the current payload. A request parameter of `ignoreEmptyCollection` set to `true` can be used in case you do not wish to send in any attributes/relationships and also do not want them to be deleted from the system.

It is not allowed to update an already deleted tracked entity instance. Also, it is not allowed to mark a tracked entity instance as deleted via an update request. The same rules apply to enrollments and events.

Deleting a tracked entity instance

In order to delete a tracked entity instance, make a request to the URL identifying the tracked entity instance with the *DELETE* method. The URL is equal to the one above used for update.

Create and enroll tracked entity instances

It is also possible to both create (and update) a tracked entity instance and at the same time enroll into a program.

```
{
    "trackedEntity": "tracked-entity-id",
    "orgUnit": "org-unit-id",
    "attributes": [
        {
            "attribute": "attribute-id",
            "value": "attribute-value"
        }
    ],
    "enrollments": [
        {
            "orgUnit": "org-unit-id",
            "program": "program-id",
            "enrollmentDate": "2013-09-17",
            "incidentDate": "2013-09-17"
        },
        {
            "orgUnit": "org-unit-id",
            "program": "program-id",
            "enrollmentDate": "2013-09-17",
            "incidentDate": "2013-09-17"
        }
    ]
}
```

You would send this to the server as you would normally when creating or updating a new tracked entity instance.

```
curl -X POST -d @tei.json -H "Content-Type: application/json"
  -u user:pass "http://server/api/33/trackedEntityInstances"
```

Complete example of payload including: tracked entity instance, enrollment and event

It is also possible to create (and update) a tracked entity instance, at the same time enroll into a program and create an event.

```
{
    "trackedEntityType": "nEenWmSyUEp",
    "orgUnit": "DiszpKrYNg8",
    "attributes": [
        {
            "attribute": "w75KJ2mc4zz",
            "value": "Joe"
        },
        {
            "attribute": "zDhUuAYrxNC",
            "value": "Rufus"
```

```
        },
        {
            "attribute": "cejWyOfXge6",
            "value": "Male"
        }
    ],
    "enrollments": [
        {
            "orgUnit": "DiszpKrYNg8",
            "program": "ur1Edk5Oe2n",
            "enrollmentDate": "2017-09-15",
            "incidentDate": "2017-09-15",
            "events": [
                {
                    "program": "ur1Edk5Oe2n",
                    "orgUnit": "DiszpKrYNg8",
                    "eventDate": "2017-10-17",
                    "status": "COMPLETED",
                    "storedBy": "admin",
                    "programStage": "EPEcjy3FWmI",
                    "coordinate": {
                        "latitude": "59.8",
                        "longitude": "10.9"
                    },
                    "dataValues": [
                        {
                            "dataElement": "qrur9Dvnyt5",
                            "value": "22"
                        },
                        {
                            "dataElement": "oZg33kd9taw",
                            "value": "Male"
                        }
                    ]
                },
                {
                    "program": "ur1Edk5Oe2n",
                    "orgUnit": "DiszpKrYNg8",
                    "eventDate": "2017-10-17",
                    "status": "COMPLETED",
                    "storedBy": "admin",
                    "programStage": "EPEcjy3FWmI",
                    "coordinate": {
                        "latitude": "59.8",
                        "longitude": "10.9"
                    },
                    "dataValues": [
                        {
                            "dataElement": "qrur9Dvnyt5",
                            "value": "26"
                        },
                        {
                            "dataElement": "oZg33kd9taw",
                            "value": "Female"
                        }
                    ]
                }
            ]
        }
    ]
}
```

You would send this to the server as you would normally when creating or updating a new tracked entity instance.

```
curl -X POST -d @tei.json -H "Content-Type: application/json"
  -u user:pass "http://server/api/33/trackedEntityInstances"
```

Generated tracked entity instance attributes

Tracked entity instance attributes that are using automatic generation of unique values have three endpoints that are used by apps. The endpoints are all used for generating and reserving values.

In 2.29 we introduced TextPattern for defining and generating these patterns. All existing patterns will be converted to a valid TextPattern when upgrading to 2.29.

> **Note**
>
> As of 2.29, all these endpoints will require you to include any variables reported by the `requiredValues` endpoint listed as required. Existing patterns, consisting of only #, will be upgraded to the new TextPattern syntax `RANDOM(<old-pattern>)`. The RANDOM segment of the TextPattern is not a required variable, so this endpoint will work as before for patterns defined before 2.29.

Finding required values

A TextPattern can contain variables that change based on different factors. Some of these factors will be unknown to the server, so the values for these variables have to be supplied when generating and reserving values.

This endpoint will return a map of required and optional values, that the server will inject into the TextPattern when generating new values. Required variables have to be supplied for the generation, but optional variables should only be supplied if you know what you are doing.

```
GET /api/33/trackedEntityAttributes/Gs1ICEQTPlG/requiredValues
```

```
{
    "REQUIRED": ["ORG_UNIT_CODE"],
    "OPTIONAL": ["RANDOM"]
}
```

Generate value endpoint

Online web apps and other clients that want to generate a value that will be used right away can use the simple generate endpoint. This endpoint will generate a value that is guaranteed to be unique at the time of generation. The value is also guaranteed not to be reserved. As of 2.29, this endpoint will also reserve the value generated for 3 days.

If your TextPattern includes required values, you can pass them as parameters like the example below:

The expiration time can also be overridden at the time of generation, by adding the ?`expiration=<number-of-days>` to the request.

```
GET /api/33/trackedEntityAttributes/Gs1ICEQTPlG/generate?ORG_UNIT_CODE=OSLO
```

```
{
    "ownerObject": "TRACKEDENTITYATTRIBUTE",
    "ownerUid": "Gs1ICEQTPlG",
    "key": "RANDOM(X)-OSL",
    "value": "C-OSL",
    "created": "2018-03-02T12:01:36.680",
    "expiryDate": "2018-03-05T12:01:36.678"
}
```

Generate and reserve value endpoint

The generate and reserve endpoint is used by offline clients that need to be able to register tracked entities with unique ids. They will reserve a number of unique ids that this device will then use when registering new tracked entity instances. The endpoint is called to retrieve a number of tracked entity instance reserved values. An optional parameter numberToReserve specifies how many ids to generate (default is 1).

If your TextPattern includes required values, you can pass them as parameters like the example below:

Similar to the /generate endpoint, this endpoint can also specify the expiration time in the same way. By adding the `?expiration=<number-of-days>` you can override the default 60 days.

```
GET /api/33/trackedEntityAttributes/Gs1ICEQTPlG/generateAndReserve?
numberToReserve=3&ORG_UNIT_CODE=OSLO
```

```
[
    {
        "ownerObject": "TRACKEDENTITYATTRIBUTE",
        "ownerUid": "Gs1ICEQTPlG",
        "key": "RANDOM(X)-OSL",
        "value": "B-OSL",
        "created": "2018-03-02T13:22:35.175",
        "expiryDate": "2018-05-01T13:22:35.174"
    },
    {
        "ownerObject": "TRACKEDENTITYATTRIBUTE",
        "ownerUid": "Gs1ICEQTPlG",
        "key": "RANDOM(X)-OSL",
        "value": "Q-OSL",
        "created": "2018-03-02T13:22:35.175",
        "expiryDate": "2018-05-01T13:22:35.174"
    },
    {
        "ownerObject": "TRACKEDENTITYATTRIBUTE",
        "ownerUid": "Gs1ICEQTPlG",
        "key": "RANDOM(X)-OSL",
        "value": "S-OSL",
        "created": "2018-03-02T13:22:35.175",
        "expiryDate": "2018-05-01T13:22:35.174"
```

```
    }
]
```

Reserved values

Reserved values are currently not accessible through the api, however, they are returned by the `generate` and `generateAndReserve` endpoints. The following table explains the properties of the reserved value object:

Reserved values

Property	Description
ownerObject	The metadata type referenced when generating and reserving the value. Currently only TRACKEDENTITYATTRIBUTE is supported.
ownerUid	The uid of the metadata object referenced when generating and reserving the value.
key	A partially generated value where generated segments are not yet added.
value	The fully resolved value reserved. This is the value you send to the server when storing data.
created	The timestamp when the reservation was made
expiryDate	The timestamp when the reservation will no longer be reserved

Expired reservations are removed daily. If a pattern changes, values that were already reserved will be accepted when storing data, even if they don't match the new pattern, as long as the reservation has not expired.

Image attributes

Working with image attributes is a lot like working with file data values. The value of an attribute with the image value type is the id of the associated file resource. A GET request to the `/api/trackedEntityInstances/<entityId>/<attributeId>/image` endpoint will return the actual image. The optional height and width parameters can be used to specify the dimensions of the image.

```
curl "http://server/api/33/trackedEntityInstances/ZRyCnJ1qUXS/zDhUuAYrxNC/image?
height=200&width=200"
  > image.jpg
```

The API also supports a *dimension* parameter. It can take three possible values: `small` (254x254), `medium` (512x512), `large` (1024x1024) or `original`. Image type attributes will be stored in pre-generated sizes and will be furnished upon request based on the value of the `dimension` parameter.

```
curl "http://server/api/33/trackedEntityInstances/ZRyCnJ1qUXS/zDhUuAYrxNC/image?dimension=medium"
```

Tracked entity instance query

To query for tracked entity instances you can interact with the `/api/trackedEntityInstances` resource.

```
/api/33/trackedEntityInstances
```

Request syntax

Tracked entity instances query parameters

Query parameter	Description
filter	Attributes to use as a filter for the query. Param can be repeated any number of times. Filters can be applied to a dimension on the format <attribute-id>:<operator>:<filter>[:<operator>:<filter>]. Filter values are case-insensitive and can be repeated together with operator any number of times. Operators can be EQ | GT | GE | LT | LE | NE | LIKE | IN.
ou	Organisation unit identifiers, separated by ";".
ouMode	The mode of selecting organisation units, can be SELECTED | CHILDREN | DESCENDANTS | ACCESSIBLE | CAPTURE | ALL. Default is SELECTED, which refers to the selected selected organisation units only. See table below for explanations.
program	Program identifier. Restricts instances to being enrolled in the given program.
programStatus	Status of the instance for the given program. Can be ACTIVE | COMPLETED | CANCELLED.
followUp	Follow up status of the instance for the given program. Can be true | false or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
page	The page number. Default page is 1.
pageSize	The page size. Default size is 50 rows per page.
totalPages	Indicates whether to include the total number of pages in the paging response (implies higher response time).
skipPaging	Indicates whether paging should be ignored and all rows should be returned.
lastUpdatedStartDate	Filter for events which were updated after this date. Cannot be used together with *lastUpdatedDuration*.
lastUpdatedEndDate	Filter for events which were updated up until this date. Cannot be used together with *lastUpdatedDuration*.
lastUpdatedDuration	Include only items which are updated within the given duration. The format is , where the supported time units are “d” (days), “h” (hours), “m” (minutes) and “s” (seconds). Cannot be used together with *lastUpdatedStartDate* and/or *lastUpdatedEndDate*.
assignedUserMode	Restricts result to tei with events assigned based on the assigned user selection mode, can be CURRENT | PROVIDED | NONE | ANY.
assignedUser	Filter the result down to a limited set of teis with events that are assigned to the given user IDs by using *assignedUser=id1;id2*.This parameter will be considered only if assignedUserMode is either PROVIDED or null. The API will error out, if for example, assignedUserMode=CURRENT and assignedUser=someId

Query parameter	Description
trackedEntityInstance	Filter the result down to a limited set of teis using explicit uids of the tracked entity instances by using *trackedEntityInstance=id1;id2*. This parameter will at the very least create the outer boundary of the results, forming the list of all teis using the uids provided. If other parameters/filters from this table are used, they will further limit the results from the explicit outer boundary.

The available organisation unit selection modes are explained in the following table.

Organisation unit selection modes

Mode	Description
SELECTED	Organisation units defined in the request.
CHILDREN	The selected organisation units and the immediate children, i.e. the organisation units at the level below.
DESCENDANTS	The selected organisation units and all children, i.e. all organisation units in the sub-hierarchy.
ACCESSIBLE	The data view organisation units associated with the current user and all children, i.e. all organisation units in the sub-hierarchy. Will fall back to data capture organisation units associated with the current user if the former is not defined.
CAPTURE	The data capture organisation units associated with the current user and all children, i.e. all organisation units in the sub-hierarchy.
ALL	All organisation units in the system. Requires the ALL authority.

The query is case insensitive. The following rules apply to the query parameters.

- At least one organisation unit must be specified using the *ou* parameter (one or many), or *ouMode=ALL* must be specified.
- Only one of the *program* and *trackedEntity* parameters can be specified (zero or one).
- If *programStatus* is specified then *program* must also be specified.
- If *followUp* is specified then *program* must also be specified.
- If *programStartDate* or *programEndDate* is specified then *program* must also be specified.
- Filter items can only be specified once.

A query for all instances associated with a specific organisation unit can look like this:

```
/api/33/trackedEntityInstances.json?ou=DiszpKrYNg8
```

To query for instances using one attribute with a filter and one attribute without a filter, with one organisation unit using the descendant organisation unit query mode:

```
/api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A
  &filter=AMpUYgxuCaE&ou=DiszpKrYNg8;yMCshbaVExv
```

A query for instances where one attribute is included in the response and one attribute is used as a filter:

```
/api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A
  &filter=AMpUYgxuCaE:LIKE:Road&ou=DiszpKrYNg8
```

A query where multiple operand and filters are specified for a filter item:

```
api/33/trackedEntityInstances.json?ou=DiszpKrYNg8&program=ur1Edk5Oe2n
  &filter=lw1SqmMlnfh:GT:150:LT:190
```

To query on an attribute using multiple values in an *IN* filter:

```
api/33/trackedEntityInstances.json?ou=DiszpKrYNg8
  &filter=dv3nChNSIxy:IN:Scott;Jimmy;Santiago
```

To constrain the response to instances which are part of a specific program you can include a program query parameter:

```
api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
  &ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
  &program=ur1Edk5Oe2n&programStartDate=2013-01-01&programEndDate=2013-09-01
```

To constrain the response to instances of a specific tracked entity you can include a tracked entity query parameter:

```
api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
  &ouMode=DESCENDANTS&trackedEntity=cyl5vuJ5ETQ
```

By default the instances are returned in pages of size 50, to change this you can use the page and pageSize query parameters:

```
api/33/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
  &ouMode=DESCENDANTS&page=2&pageSize=3
```

You can use a range of operators for the filtering:

Filter operators

Operator	Description
EQ	Equal to
GT	Greater than
GE	Greater than or equal to

Operator	Description
LT	Less than
LE	Less than or equal to
NE	Not equal to
LIKE	Like (free text match)
IN	Equal to one of multiple values separated by ";"

Response format

This resource supports JSON, JSONP, XLS and CSV resource representations.

- json (application/json)
- jsonp (application/javascript)
- xml (application/xml)

The response in JSON/XML is in object format and can look like the following. Please note that field filtering is supported, so if you want a full view, you might want to add `fields=*` to the query:


```
{
    "trackedEntityInstances": [
        {
            "lastUpdated": "2014-03-28 12:27:52.399",
            "trackedEntity": "cyl5vuJ5ETQ",
            "created": "2014-03-26 15:40:19.997",
            "orgUnit": "ueuQlqb8ccl",
            "trackedEntityInstance": "tphfdyIiVL6",
            "relationships": [],
            "attributes": [
                {
                    "displayName": "Address",
                    "attribute": "AMpUYgxuCaE",
                    "type": "string",
                    "value": "2033 Akasia St"
                },
                {
                    "displayName": "TB number",
                    "attribute": "ruQQnf6rswq",
                    "type": "string",
                    "value": "1Z 989 408 56 9356 521 9"
                },
                {
                    "displayName": "Weight in kg",
                    "attribute": "OvY4VVhSDeJ",
                    "type": "number",
                    "value": "68.1"
                },
                {
                    "displayName": "Email",
                    "attribute": "NDXw0cluzSw",
                    "type": "string",
                    "value": "LiyaEfrem@armyspy.com"
                },
                {
                    "displayName": "Gender",
                    "attribute": "cejWyOfXge6",
```

```
            "type": "optionSet",
            "value": "Female"
        },
        {
            "displayName": "Phone number",
            "attribute": "P2cwLGskgxn",
            "type": "phoneNumber",
            "value": "085 813 9447"
        },
        {
            "displayName": "First name",
            "attribute": "dv3nChNSIxy",
            "type": "string",
            "value": "Liya"
        },
        {
            "displayName": "Last name",
            "attribute": "hwlRTFIFSUq",
            "type": "string",
            "value": "Efrem"
        },
        {
            "code": "Height in cm",
            "displayName": "Height in cm",
            "attribute": "lw1SqmMlnfh",
            "type": "number",
            "value": "164"
        },
        {
            "code": "City",
            "displayName": "City",
            "attribute": "VUvgVao8Y5z",
            "type": "string",
            "value": "Kranskop"
        },
        {
            "code": "State",
            "displayName": "State",
            "attribute": "GUOBQt5K2WI",
            "type": "number",
            "value": "KwaZulu-Natal"
        },
        {
            "code": "Zip code",
            "displayName": "Zip code",
            "attribute": "n9nUvfpTsxQ",
            "type": "number",
            "value": "3282"
        },
        {
            "code": "National identifier",
            "displayName": "National identifier",
            "attribute": "AuPLng5hLbE",
            "type": "string",
            "value": "465700042"
        },
        {
            "code": "Blood type",
            "displayName": "Blood type",
            "attribute": "H9IlTX2X6SL",
            "type": "string",
            "value": "B-"
        },
```

```
                {
                    "code": "Latitude",
                    "displayName": "Latitude",
                    "attribute": "Qo571yj6Zcn",
                    "type": "string",
                    "value": "-30.659626"
                },
                {
                    "code": "Longitude",
                    "displayName": "Longitude",
                    "attribute": "RG7uGl4w5Jq",
                    "type": "string",
                    "value": "26.916172"
                }
            ]
        }
    ]
}
```

Tracked entity instance grid query

To query for tracked entity instances you can interact with the */api/trackedEntityInstances/grid* resource. There are two types of queries: One where a *query* query parameter and optionally *attribute* parameters are defined, and one where *attribute* and *filter* parameters are defined. This endpoint uses a more compact "grid" format, and is an alternative to the query in the previous section.

```
/api/33/trackedEntityInstances/query
```

Request syntax

Tracked entity instances query parameters

Query parameter	Description
query	Query string. Attribute query parameter can be used to define which attributes to include in the response. If no attributes but a program is defined, the attributes from the program will be used. If no program is defined, all attributes will be used. There are two formats. The first is a plan query string. The second is on the format <operator>:<query>. Operators can be EQ | LIKE. EQ implies exact matches on words, LIKE implies partial matches on words. The query will be split on space, where each word will form a logical AND query.
attribute	Attributes to be included in the response. Can also be used as a filter for the query. Param can be repeated any number of times. Filters can be applied to a dimension on the format <attribute-id>:<operator>:<filter>[:<operator>:<filter>]. Filter values are case-insensitive and can be repeated together with operator any number of times. Operators can be EQ | GT | GE | LT | LE | NE | LIKE | IN. Filters can be omitted in order to simply include the attribute in the response without any constraints.
filter	Attributes to use as a filter for the query. Param can be repeated any number of times. Filters can be applied to a dimension on the format <attribute-id>:<operator>:<filter>[:<operator>:<filter>]. Filter values are case-insensitive and can be repeated together with operator any number of times. Operators can be EQ | GT | GE | LT | LE | NE | LIKE | IN.
ou	Organisation unit identifiers, separated by ";".

Query parameter	Description
ouMode	The mode of selecting organisation units, can be SELECTED | CHILDREN | DESCENDANTS | ACCESSIBLE | ALL. Default is SELECTED, which refers to the selected organisation units only. See table below for explanations.
program	Program identifier. Restricts instances to being enrolled in the given program.
programStatus	Status of the instance for the given program. Can be ACTIVE | COMPLETED | CANCELLED.
followUp	Follow up status of the instance for the given program. Can be true | false or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
eventStatus	Status of any event associated with the given program and the tracked entity instance. Can be ACTIVE | COMPLETED | VISITED | SCHEDULED | OVERDUE | SKIPPED.
eventStartDate	Start date of event associated with the given program and event status.
eventEndDate	End date of event associated with the given program and event status.
programStage	The programStage for which the event related filters should be applied to. If not provided all stages will be considered.
skipMeta	Indicates whether meta data for the response should be included.
page	The page number. Default page is 1.
pageSize	The page size. Default size is 50 rows per page.
totalPages	Indicates whether to include the total number of pages in the paging response (implies higher response time).
skipPaging	Indicates whether paging should be ignored and all rows should be returned.
assignedUserMode	Restricts result to tei with events assigned based on the assigned user selection mode, can be CURRENT | PROVIDED | NONE | ANY.
assignedUser	Filter the result down to a limited set of teis with events that are assigned to the given user IDs by using *assignedUser=id1;id2*.This parameter will be considered only if assignedUserMode is either PROVIDED or null. The API will error out, if for example, assignedUserMode=CURRENT and assignedUser=someId
trackedEntityInstance	Filter the result down to a limited set of teis using explicit uids of the tracked entity instances by using *trackedEntityInstance=id1;id2*. This parameter will at the very least create the outer boundary of the results, forming the list of all teis using the uids provided. If other parameters/filters from this table are used, they will further limit the results from the explicit outer boundary.

The available organisation unit selection modes are explained in the following table.

Organisation unit selection modes

Mode	Description
SELECTED	Organisation units defined in the request.

Mode	Description
CHILDREN	Immediate children, i.e. only the first level below, of the organisation units defined in the request.
DESCENDANTS	All children, i.e. at only levels below, e.g. including children of children, of the organisation units defined in the request.
ACCESSIBLE	All descendants of the data view organisation units associated with the current user. Will fall back to data capture organisation units associated with the current user if the former is not defined.
CAPTURE	The data capture organisation units associated with the current user and all children, i.e. all organisation units in the sub-hierarchy.
ALL	All organisation units in the system. Requires authority.

Note that you can specify "attribute" with filters or directly using the "filter" params for constraining the instances to return.

Certain rules apply to which attributes are returned.

- If "query" is specified without any attributes or program, then all attributes that are marked as "Display in List without Program" is included in the response.
- If program is specified, all the attributes linked to the program will be included in the response.
- If tracked entity type is specified, then all tracked entity type attributes will be included in the response.

You can specify queries with words separated by space - in that situation the system will query for each word independently and return records where each word is contained in any attribute. A query item can be specified once as an attribute and once as a filter if needed. The query is case insensitive. The following rules apply to the query parameters.

- At least one organisation unit must be specified using the *ou* parameter (one or many), or *ouMode=ALL* must be specified.
- Only one of the *program* and *trackedEntity* parameters can be specified (zero or one).
- If *programStatus* is specified then *program* must also be specified.
- If *followUp* is specified then *program* must also be specified.
- If *programStartDate* or *programEndDate* is specified then *program* must also be specified.
- If *eventStatus* is specified then *eventStartDate* and *eventEndDate* must also be specified.
- A query cannot be specified together with filters.
- Attribute items can only be specified once.
- Filter items can only be specified once.

A query for all instances associated with a specific organisation unit can look like this:

```
/api/33/trackedEntityInstances/query.json?ou=DiszpKrYNg8
```

A query on all attributes for a specific value and organisation unit, using an exact word match:

```
/api/33/trackedEntityInstances/query.json?query=scott&ou=DiszpKrYNg8
```

A query on all attributes for a specific value, using a partial word match:

```
/api/33/trackedEntityInstances/query.json?query=LIKE:scott&ou=DiszpKrYNg8
```

You can query on multiple words separated by the URL character for space which is %20, will use a logical AND query for each word:

```
/api/33/trackedEntityInstances/query.json?query=isabel%20may&ou=DiszpKrYNg8
```

A query where the attributes to include in the response are specified:

```
/api/33/trackedEntityInstances/query.json?query=isabel
  &attribute=dv3nChNSIxy&attribute=AMpUYgxuCaE&ou=DiszpKrYNg8
```

To query for instances using one attribute with a filter and one attribute without a filter, with one organisation unit using the descendants organisation unit query mode:

```
/api/33/trackedEntityInstances/query.json?attribute=zHXD5Ve1Efw:EQ:A
  &attribute=AMpUYgxuCaE&ou=DiszpKrYNg8;yMCshbaVExv
```

A query for instances where one attribute is included in the response and one attribute is used as a filter:

```
/api/33/trackedEntityInstances/query.json?attribute=zHXD5Ve1Efw:EQ:A
  &filter=AMpUYgxuCaE:LIKE:Road&ou=DiszpKrYNg8
```

A query where multiple operand and filters are specified for a filter item:

```
/api/33/trackedEntityInstances/query.json?ou=DiszpKrYNg8&program=ur1Edk5Oe2n
  &filter=lw1SqmMlnfh:GT:150:LT:190
```

To query on an attribute using multiple values in an IN filter:

```
/api/33/trackedEntityInstances/query.json?ou=DiszpKrYNg8
  &attribute=dv3nChNSIxy:IN:Scott;Jimmy;Santiago
```

To constrain the response to instances which are part of a specific program you can include a program query parameter:

```
/api/33/trackedEntityInstances/query.json?filter=zHXD5Ve1Efw:EQ:A
  &ou=O6uvpzGd5pu&ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
/api/33/trackedEntityInstances/query.json?filter=zHXD5Ve1Efw:EQ:A
  &ou=O6uvpzGd5pu&program=ur1Edk5Oe2n&programStartDate=2013-01-01
  &programEndDate=2013-09-01
```

To constrain the response to instances of a specific tracked entity you can include a tracked entity query parameter:

```
/api/33/trackedEntityInstances/query.json?attribute=zHXD5Ve1Efw:EQ:A
  &ou=O6uvpzGd5pu&ouMode=DESCENDANTS&trackedEntity=cyl5vuJ5ETQ
```

By default the instances are returned in pages of size 50, to change this you can use the page and pageSize query parameters:

```
/api/33/trackedEntityInstances/query.json?attribute=zHXD5Ve1Efw:EQ:A
  &ou=O6uvpzGd5pu&ouMode=DESCENDANTS&page=2&pageSize=3
```

To query for instances which have events of a given status within a given time span:

```
/api/33/trackedEntityInstances/query.json?ou=O6uvpzGd5pu
  &program=ur1Edk5Oe2n&eventStatus=LATE_VISIT
  &eventStartDate=2014-01-01&eventEndDate=2014-09-01
```

You can use a range of operators for the filtering:

Filter operators

Operator	Description
EQ	Equal to
GT	Greater than
GE	Greater than or equal to
LT	Less than
LE	Less than or equal to
NE	Not equal to
LIKE	Like (free text match)
IN	Equal to one of multiple values separated by ";"

Response format

This resource supports JSON, JSONP, XLS and CSV resource representations.

- json (application/json)
- jsonp (application/javascript)
- xml (application/xml)
- csv (application/csv)
- xls (application/vnd.ms-excel)

The response in JSON comes is in a tabular format and can look like the following. The *headers* section describes the content of each column. The instance, created, last updated, org unit and tracked entity columns are always present. The following columns correspond to attributes specified in the query. The *rows* section contains one row per instance.


```
{
    "headers": [
        {
            "name": "instance",
            "column": "Instance",
            "type": "java.lang.String"
        },
        {
            "name": "created",
            "column": "Created",
            "type": "java.lang.String"
        },
        {
            "name": "lastupdated",
            "column": "Last updated",
            "type": "java.lang.String"
        },
        {
            "name": "ou",
            "column": "Org unit",
            "type": "java.lang.String"
        },
        {
            "name": "te",
            "column": "Tracked entity",
            "type": "java.lang.String"
        },
        {
            "name": "zHXD5Ve1Efw",
            "column": "Date of birth type",
            "type": "java.lang.String"
        },
        {
            "name": "AMpUYgxuCaE",
            "column": "Address",
            "type": "java.lang.String"
        }
    ],
    "metaData": {
        "names": {
            "cyl5vuJ5ETQ": "Person"
        }
    },
    "width": 7,
    "height": 7,
    "rows": [
        [
            "yNCtJ6vhRJu",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:39:32.19",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "21 Kenyatta Road"
        ],
        [
            "fSofnQR6lAU",
```

```
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:40:19.62",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "56 Upper Road"
        ],
        [
            "X5wZwS5lgm2",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:40:31.11",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "56 Main Road"
        ],
        [
            "pCbogmlIXga",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:40:45.02",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "12 Lower Main Road"
        ],
        [
            "WnUXrY4XBMM",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:41:06.97",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "13 Main Road"
        ],
        [
            "xLNXbDs9uDF",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:42:25.66",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "14 Mombasa Road"
        ],
        [
            "foc5zag6gbE",
            "2013-09-08 21:40:28.0",
            "2014-01-09 19:42:36.93",
            "DiszpKrYNg8",
            "cyl5vuJ5ETQ",
            "A",
            "15 Upper Hill"
        ]
    ]
}
```


Tracked entity instance filters

To create, read, update and delete tracked entity instance filters you can interact with the */api/trackedEntityInstanceFilters* resource.

```
/api/33/trackedEntityInstanceFilters
```

Create and update a tracked entity instance filter definition

For creating and updating a tracked entity instance filter in the system, you will be working with the *trackedEntityInstanceFilters* resource. The tracked entity instance filter definitions are used in the Tracker Capture app to display relevant predefined "Working lists" in the tracker user interface.

Payload

Payload values	Description	Example
name	Name of the filter. Required.	
description	A description of the filter.	
sortOrder	The sort order of the filter. Used in Tracker Capture to order the filters in the program dashboard.	
style	Object containing css style.	("color": "blue", "icon": "fa fa-calendar"}
program	Object containing the id of the program. Required.	{ "id" : "uy2gU8kTjF"}
enrollmentStatus	The TEIs enrollment status. Can be none(any enrollmentstatus) or ACTIVE|COMPLETED| CANCELED	
followup	When this parameter is true, the filter only returns TEIs that have an enrollment with status followup.	
enrollmentCreatedPeriod	Period object containing a period in which the enrollment must be created. See *Period* definition table below.	{ "periodFrom": -15, "periodTo": 15}
eventFilters	A list of eventFilters. See *Event filters* definition table below.	[{"programStage": "eaDH9089uMp", "eventStatus": "OVERDUE", "eventCreatedPeriod": {"periodFrom": -15, "periodTo": 15}}]

Event filters definition

programStage	Which programStage the TEI needs an event in to be returned.	"eaDH9089uMp"
eventStatus	The events status. Can be none(any event status) or ACTIVE|COMPLETED| SCHEDULED|OVERDUE	ACTIVE

eventCreatedPeriod	Period object containing a period in which the event must be created. See *Period* definition below.	{ "periodFrom": -15, "periodTo": 15}
assignedUserMode	To specify the assigned user selection mode for events. Possible values are CURRENT (events assigned to current user)| PROVIDED (events assigned to users provided in "assignedUsers" list) | NONE (events assigned to no one) | ANY (events assigned to anyone). If PROVIDED (or null), non-empty assignedUsers in the payload will be considered.	"assignedUserMode": "PROVIDED"
assignedUsers	To specify a list of assigned users for events. To be used along with PROVIDED assignedUserMode above.	"assignedUsers": ["a3kGcGDCuk7", "a3kGcGDCuk8"]

Period definition

periodFrom	Number of days from current day. Can be positive or negative integer.	-15
periodTo	Number of days from current day. Must be bigger than periodFrom. Can be positive or negative integer.	15

Tracked entity instance filters query

To query for tracked entity instance filters in the system, you can interact with the */api/trackedEntityInstanceFilters* resource.

Tracked entity instance filters query parameters

Query parameter	Description
program	Program identifier. Restricts filters to the given program.

Enrollment management

Enrollments have full CRUD support in the API. Together with the API for tracked entity instances most operations needed for working with tracked entity instances and programs are supported.

```
/api/33/enrollments
```

Enrolling a tracked entity instance into a program

For enrolling persons into a program, you will need to first get the identifier of the person from the *trackedEntityInstances* resource. Then, you will need to get the program identifier from the *programs* resource. A template payload can be seen below:

```
{
    "trackedEntityInstance": "ZRyCnJ1qUXS",
    "orgUnit": "ImspTQPwCqd",
    "program": "S8uo8AlvYMz",
    "enrollmentDate": "2013-09-17",
    "incidentDate": "2013-09-17"
}
```

This payload should be used in a *POST* request to the enrollments resource identified by the following URL:

```
/api/33/enrollments
```

For cancelling or completing an enrollment, you can make a *PUT* request to the `enrollments` resource, including the identifier and the action you want to perform. For cancelling an enrollment for a tracked entity instance:

```
/api/33/enrollments/<enrollment-id>/cancelled
```

For completing an enrollment for a tracked entity instance you can make a *PUT* request to the following URL:

```
/api/33/enrollments/<enrollment-id>/completed
```

For deleting an enrollment, you can make a *DELETE* request to the following URL:

```
/api/33/enrollments/<enrollment-id>
```

Enrollment instance query

To query for enrollments you can interact with the */api/enrollments* resource.

```
/api/33/enrollments
```

Request syntax

Enrollment query parameters

Query parameter	Description
ou	Organisation unit identifiers, separated by ";".

Query parameter	Description
ouMode	The mode of selecting organisation units, can be SELECTED | CHILDREN | DESCENDANTS | ACCESSIBLE | CAPTURE | ALL. Default is SELECTED, which refers to the selected organisation units only. See table below for explanations.
program	Program identifier. Restricts instances to being enrolled in the given program.
programStatus	Status of the instance for the given program. Can be ACTIVE | COMPLETED | CANCELLED.
followUp	Follow up status of the instance for the given program. Can be true | false or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
lastUpdatedDuration	Include only items which are updated within the given duration. The format is , where the supported time units are “d” (days), “h” (hours), “m” (minutes) and “s” (seconds).
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
trackedEntityInstance	Tracked entity instance identifier. Should not be used together with trackedEntity.
page	The page number. Default page is 1.
pageSize	The page size. Default size is 50 rows per page.
totalPages	Indicates whether to include the total number of pages in the paging response (implies higher response time).
skipPaging	Indicates whether paging should be ignored and all rows should be returned.
includeDeleted	Indicates whether to include soft deleted enrollments or not. It is false by default.

The available organisation unit selection modes are explained in the following table.

Organisation unit selection modes

Mode	Description
SELECTED	Organisation units defined in the request (default).
CHILDREN	Immediate children, i.e. only the first level below, of the organisation units defined in the request.
DESCENDANTS	All children, i.e. at only levels below, e.g. including children of children, of the organisation units defined in the request.
ACCESSIBLE	All descendants of the data view organisation units associated with the current user. Will fall back to data capture organisation units associated with the current user if the former is not defined.
ALL	All organisation units in the system. Requires authority.

The query is case insensitive. The following rules apply to the query parameters.

- At least one organisation unit must be specified using the *ou* parameter (one or many), or *ouMode=ALL* must be specified.
- Only one of the *program* and *trackedEntity* parameters can be specified (zero or one).
- If *programStatus* is specified then *program* must also be specified.

If *followUp* is specified then *program* must also be specified.

- If *programStartDate* or *programEndDate* is specified then *program* must also be specified.

A query for all enrollments associated with a specific organisation unit can look like this:

```
/api/33/enrollments.json?ou=DiszpKrYNg8
```

To constrain the response to enrollments which are part of a specific program you can include a program query parameter:

```
/api/33/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
/api/33/enrollments.json?&ou=O6uvpzGd5pu&program=ur1Edk5Oe2n
  &programStartDate=2013-01-01&programEndDate=2013-09-01
```

To constrain the response to enrollments of a specific tracked entity you can include a tracked entity query parameter:

```
/api/33/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&trackedEntity=cyl5vuJ5ETQ
```

To constrain the response to enrollments of a specific tracked entity instance you can include a tracked entity instance query parameter, in this case we have restricted it to available enrollments viewable for current user:

```
/api/33/enrollments.json?ouMode=ACCESSIBLE&trackedEntityInstance=tphfdyIiVL6
```

By default the enrollments are returned in pages of size 50, to change this you can use the page and pageSize query parameters:

```
/api/33/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&page=2&pageSize=3
```

Response format

This resource supports JSON, JSONP, XLS and CSV resource representations.

- json (application/json)
- jsonp (application/javascript)
- xml (application/xml)

The response in JSON/XML is in object format and can look like the following. Please note that field filtering is supported, so if you want a full view, you might want to add `fields=*` to the query:

```
{
    "enrollments": [
        {
            "lastUpdated": "2014-03-28T05:27:48.512+0000",
            "trackedEntity": "cyl5vuJ5ETQ",
            "created": "2014-03-28T05:27:48.500+0000",
            "orgUnit": "DiszpKrYNg8",
            "program": "ur1Edk5Oe2n",
            "enrollment": "HLFOK0XThjr",
            "trackedEntityInstance": "qv0j4JBXQX0",
            "followup": false,
            "enrollmentDate": "2013-05-23T05:27:48.490+0000",
            "incidentDate": "2013-05-10T05:27:48.490+0000",
            "status": "ACTIVE"
        }
    ]
}
```

Events

This section is about sending and reading events.

```
/api/33/events
```

Sending events

DHIS2 supports three kinds of events: single events with no registration (also referred to as anonymous events), single event with registration and multiple events with registration. Registration implies that the data is linked to a tracked entity instance which is identified using some sort of identifier.

To send events to DHIS2 you must interact with the *events* resource. The approach to sending events is similar to sending aggregate data values. You will need a *program* which can be looked up using the *programs* resource, an *orgUnit* which can be looked up using the *organisationUnits* resource, and a list of valid data element identifiers which can be looked up using the *dataElements* resource. For events with registration, a *tracked entity instance* identifier is required, read about how to get this in the section about the *trackedEntityInstances* resource. For sending events to programs with multiple stages, you will need to also include the *programStage* identifier, the identifiers for programStages can be found in the *programStages* resource.

A simple single event with no registration example payload in XML format where we send events from the "Inpatient morbidity and mortality" program for the "Ngelehun CHC" facility in the demo database can be seen below:

```
<?xml version="1.0" encoding="utf-8"?>
<event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8"
  eventDate="2013-05-17" status="COMPLETED" storedBy="admin">
  <coordinate latitude="59.8" longitude="10.9" />
  <dataValues>
    <dataValue dataElement="qrur9Dvnyt5" value="22" />
    <dataValue dataElement="oZg33kd9taw" value="Male" />
    <dataValue dataElement="msodh3rEMJa" value="2013-05-18" />
  </dataValues>
</event>
```

To perform some testing we can save the XML payload as a file called*event.xml* and send it as a POST request to the events resource in the API using curl with the following command:

```
curl -d @event.xml "https://play.dhis2.org/demo/api/33/events"
  -H "Content-Type:application/xml" -u admin:district
```

The same payload in JSON format looks like this:

```
{
    "program": "eBAyeGv0exc",
    "orgUnit": "DiszpKrYNg8",
    "eventDate": "2013-05-17",
    "status": "COMPLETED",
    "completedDate": "2013-05-18",
    "storedBy": "admin",
    "coordinate": {
        "latitude": 59.8,
        "longitude": 10.9
    },
    "dataValues": [
        {
            "dataElement": "qrur9Dvnyt5",
            "value": "22"
        },
        {
            "dataElement": "oZg33kd9taw",
            "value": "Male"
        },
        {
            "dataElement": "msodh3rEMJa",
            "value": "2013-05-18"
        }
    ]
}
```

To send this you can save it to a file called *event.json* and use curl like this:

```
curl -d @event.json "localhost/api/33/events" -H "Content-Type:application/json"
  -u admin:district
```

We also support sending multiple events at the same time. A payload in XML format might look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<events>
  <event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8"
    eventDate="2013-05-17" status="COMPLETED" storedBy="admin">
    <coordinate latitude="59.8" longitude="10.9" />
    <dataValues>
      <dataValue dataElement="qrur9Dvnyt5" value="22" />
      <dataValue dataElement="oZg33kd9taw" value="Male" />
    </dataValues>
  </event>
  <event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8"
    eventDate="2013-05-17" status="COMPLETED" storedBy="admin">
    <coordinate latitude="59.8" longitude="10.9" />
```

```
    <dataValues>
      <dataValue dataElement="qrur9Dvnyt5" value="26" />
      <dataValue dataElement="oZg33kd9taw" value="Female" />
    </dataValues>
  </event>
</events>
```

You will receive an import summary with the response which can be inspected in order to get information about the outcome of the request, like how many values were imported successfully. The payload in JSON format looks like this:

```
{
    "events": [
        {
            "program": "eBAyeGv0exc",
            "orgUnit": "DiszpKrYNg8",
            "eventDate": "2013-05-17",
            "status": "COMPLETED",
            "storedBy": "admin",
            "coordinate": {
                "latitude": "59.8",
                "longitude": "10.9"
            },
            "dataValues": [
                {
                    "dataElement": "qrur9Dvnyt5",
                    "value": "22"
                },
                {
                    "dataElement": "oZg33kd9taw",
                    "value": "Male"
                }
            ]
        },
        {
            "program": "eBAyeGv0exc",
            "orgUnit": "DiszpKrYNg8",
            "eventDate": "2013-05-17",
            "status": "COMPLETED",
            "storedBy": "admin",
            "coordinate": {
                "latitude": "59.8",
                "longitude": "10.9"
            },
            "dataValues": [
                {
                    "dataElement": "qrur9Dvnyt5",
                    "value": "26"
                },
                {
                    "dataElement": "oZg33kd9taw",
                    "value": "Female"
                }
            ]
        }
    ]
}
```

You can also use GeoJson to store any kind of geometry on your event. An example payload using GeoJson instead of the former latitude and longitude properties can be seen here:

```
{
    "program": "eBAyeGv0exc",
    "orgUnit": "DiszpKrYNg8",
    "eventDate": "2013-05-17",
    "status": "COMPLETED",
    "storedBy": "admin",
    "geometry": {
        "type": "POINT",
        "coordinates": [59.8, 10.9]
    },
    "dataValues": [
        {
            "dataElement": "qrur9Dvnyt5",
            "value": "22"
        },
        {
            "dataElement": "oZg33kd9taw",
            "value": "Male"
        },
        {
            "dataElement": "msodh3rEMJa",
            "value": "2013-05-18"
        }
    ]
}
```

As part of the import summary you will also get the identifier *reference* to the event you just sent, together with a *href* element which points to the server location of this event. The table below describes the meaning of each element.

Events resource format

Parameter	Type	Required	Options (default first)	Description
program	string	true		Identifier of the single event with no registration program
orgUnit	string	true		Identifier of the organisation unit where the event took place
eventDate	date	true		The date of when the event occurred
completedDate	date	false		The date of when the event is completed. If not provided, the current date is selected as the event completed date
status	enum	false	ACTIVE | COMPLETED | VISITED | SCHEDULE | OVERDUE | SKIPPED	Whether the event is complete or not
storedBy	string	false	Defaults to current user	Who stored this event (can be username, system-name, etc)
coordinate	double	false		Refers to where the event took place geographically (latitude and longitude)
dataElement	string	true		Identifier of data element

Parameter	Type	Required	Options (default first)	Description
value	string	true		Data value or measure for this event

OrgUnit matching

By default the orgUnit parameter will match on the ID, you can also select the orgUnit id matching scheme by using the parameter orgUnitIdScheme=SCHEME, where the options are: *ID*, *UID*, *UUID*, *CODE*, and *NAME*. There is also the *ATTRIBUTE:* scheme, which matches on a *unique* metadata attribute value.

Updating events

To update an existing event, the format of the payload is the same, but the URL you are posting to must add the identifier to the end of the URL string and the request must be PUT.

The payload has to contain all, even non-modified, attributes. Attributes that were present before and are not present in the current payload any more will be removed by the system.

It is not allowed to update an already deleted event. The same applies to tracked entity instance and enrollment.

```
curl -X PUT -d @updated_event.xml "localhost/api/33/events/ID"
  -H "Content-Type: application/xml" -u admin:district
```

```
curl -X PUT -d @updated_event.json "localhost/api/33/events/ID"
  -H "Content-Type: application/json" -u admin:district
```

Deleting events

To delete an existing event, all you need is to send a DELETE request with an identifier reference to the server you are using.

```
curl -X DELETE "localhost/api/33/events/ID" -u admin:district
```

Assigning user to events

A user can be assigned to an event. This can be done by including the appropriate property in the payload when updating or creating the event.

```
  "assignedUser": "<id>"
```

The id refers to the if of the user. Only one user can be assigned to an event at a time.

User assignment must be enabled in the program stage before users can be assigned to events.

Getting events

To get an existing event you can issue a GET request including the identifier like this:

```
curl "http://localhost/api/33/events/ID" -H "Content-Type: application/xml" -u admin:district
```

Querying and reading events

This section explains how to read out the events that have been stored in the DHIS2 instance. For more advanced uses of the event data, please see the section on event analytics. The output format from the `/api/events` endpoint will match the format that is used to send events to it (which the analytics event api does not support). Both XML and JSON are supported, either through adding .json/.xml or by setting the appropriate *Accept* header. The query is paged by default and the default page size is 50 events, *field* filtering works as it does for metadata, add the *fields* parameter and include your wanted properties, i.e. *?fields=program,status*.

Events resource query parameters

Key	Type	Required	Description
program	identifier	true (if not programStage is provided)	Identifier of program
programStage	identifier	false	Identifier of program stage
programStatus	enum	false	Status of event in program, ca be ACTIVE | COMPLETED | CANCELLED
followUp	boolean	false	Whether event is considered for follow up in program, can be true | false or omitted.
trackedEntityInstance	identifier	false	Identifier of tracked entity instance
orgUnit	identifier	true	Identifier of organisation unit
ouMode	enum	false	Org unit selection mode, can be SELECTED | CHILDREN | DESCENDANTS
startDate	date	false	Only events newer than this date
endDate	date	false	Only events older than this date
status	enum	false	Status of event, can be ACTIVE | COMPLETED | VISITED | SCHEDULED | OVERDUE | SKIPPED

Key	Type	Required	Description
lastUpdatedStartDate	date	false	Filter for events which were updated after this date. Cannot be used together with *lastUpdatedDuration*.
lastUpdatedEndDate	date	false	Filter for events which were updated up until this date. Cannot be used together with *lastUpdatedDuration*.
lastUpdatedDuration	string	false	Include only items which are updated within the given duration. The format is , where the supported time units are “d” (days), “h” (hours), “m” (minutes) and “s” (seconds). Cannot be used together with *lastUpdatedStartDate* and /or *lastUpdatedEndDate*.
skipMeta	boolean	false	Exclude the meta data part of response (improves performance)
page	integer	false	Page number
pageSize	integer	false	Number of items in each page
totalPages	boolean	false	Indicates whether to include the total number of pages in the paging response.
skipPaging	boolean	false	Indicates whether to skip paging in the query and return all events.
dataElementIdScheme	string	false	Data element ID scheme to use for export, valid options are UID, CODE and ATTRIBUTE:{ID}
categoryOptionComboIdScheme	string	false	Category Option Combo ID scheme to use for export, valid options are UID, CODE and ATTRIBUTE:{ID}

Key	Type	Required	Description
orgUnitIdScheme	string	false	Organisation Unit ID scheme to use for export, valid options are UID, CODE and ATTRIBUTE:{ID}
programIdScheme	string	false	Program ID scheme to use for export, valid options are UID, CODE and ATTRIBUTE:{ID}
programStageIdScheme	string	false	Program Stage ID scheme to use for export, valid options are UID, CODE and ATTRIBUTE:{ID}
idScheme	string	false	Allows to set id scheme for data element, category option combo, orgUnit, program and program stage at once.

order	string	false	The order of which to retrieve the events from the API. Usage: order=<property>:asc/desc - Ascending order is default.
			
			Properties: event | program | programStage | enrollment | enrollmentStatus | orgUnit | orgUnitName | trackedEntityInstance | eventDate | followup | status | dueDate | storedBy | created | lastUpdated | completedBy | completedDate
			
			`order=orgUnitName:DESC`
			
			`order=lastUpdated:ASC`

event	comma delimited string	false	Filter the result down to a limited set of IDs by using *event=id1;id2*.
skipEventId	boolean	false	Skips event identifiers in the response

Key	Type	Required	Description
attributeCc (**)	string	false	Attribute category combo identifier (must be combined with *attributeCos*)
attributeCos (**)	string	false	Attribute category option identifiers, separated with ; (must be combined with *attributeCc*)
async	false | true	false	Indicates whether the import should be done asynchronous or synchronous.
includeDeleted	boolean	false	When true, soft deleted events will be included in your query result.
assignedUserMode	enum	false	Assigned user selection mode, can be CURRENT | PROVIDED | NONE | ANY.
assignedUser	comma delimited strings	false	Filter the result down to a limited set of events that are assigned to the given user IDs by using *assignedUser=id1;id2*. This parameter will be considered only if assignedUserMode is either PROVIDED or null. The API will error out, if for example, assignedUserMode=CURRENT and assignedUser=someId

> **Note**
>
> If the query contains neither `attributeCC` nor `attributeCos`, the server returns events for all attribute option combos where the user has read access.

Examples

Query for all events with children of a certain organisation unit:

```
/api/29/events.json?orgUnit=YuQRtpLP10I&ouMode=CHILDREN
```

Query for all events with all descendants of a certain organisation unit, implying all organisation units in the sub-hierarchy:

```
/api/33/events.json?orgUnit=O6uvpzGd5pu&ouMode=DESCENDANTS
```

Query for all events with a certain program and organisation unit:

```
/api/33/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc
```

Query for all events with a certain program and organisation unit, sorting by due date ascending:

```
/api/33/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc&order=dueDate
```

Query for the 10 events with the newest event date in a certain program and organisation unit - by paging and ordering by due date descending:

```
/api/33/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc
  &order=eventDate:desc&pageSize=10&page=1
```

Query for all events with a certain program and organisation unit for a specific tracked entity instance:

```
/api/33/events.json?orgUnit=DiszpKrYNg8
  &program=eBAyeGv0exc&trackedEntityInstance=gfVxE3ALA9m
```

Query for all events with a certain program and organisation unit older or equal to 2014-02-03:

```
/api/33/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc&endDate=2014-02-03
```

Query for all events with a certain program stage, organisation unit and tracked entity instance in the year 2014:

```
/api/33/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc
  &trackedEntityInstance=gfVxE3ALA9m&startDate=2014-01-01&endDate=2014-12-31
```

Query files associated with event data values. In the specific case of fetching an image file an additional parameter can be provided to fetch the image with different dimensions. If dimension is not provided, the system will return the original image. The parameter will be ignored in case of fetching non-image files e.g pdf. Possible dimension values are *small(254 x 254), medium(512 x 512), large(1024 x 1024) or original*. Any value other than those mentioned will be discarded and the original image will be returned.

```
/api/33/events/files?eventUid=hcmcWlYkg9u&dataElementUid=C0W4aFuVm4P&dimension=small
```

Retrieve events with specified Organisation unit and Program, and use *Attribute:Gq0oWTf2DtN* as identifier scheme

```
/api/events?orgUnit=DiszpKrYNg8&program=lxAQ7Zs9VYR&idScheme=Attribute:Gq0oWTf2DtN
```

Retrieve events with specified Organisation unit and Program, and use UID as identifier scheme for orgUnits, Code as identifier scheme for Program stages, and *Attribute:Gq0oWTf2DtN* as identifier scheme for the rest of the metadata with assigned attribute.

```
api/events.json?orgUnit=DiszpKrYNg8&program=lxAQ7Zs9VYR&idScheme=Attribute:Gq0oWTf2DtN
  &orgUnitIdScheme=UID&programStageIdScheme=Code
```

Event grid query

In addition to the above event query end point, there is an event grid query end point where a more compact "grid" format of events are returned. This is possible by interacting with /api/events/query.json|xml|xls|csv endpoint.

```
/api/33/events/query
```

Most of the query parameters mentioned in event querying and reading section above are valid here. However, since the grid to be returned comes with specific set of columns that apply to all rows (events), it is mandatory to specify a program stage. It is not possible to mix events from different programs or program stages in the return.

Returning events from a single program stage, also opens up for new functionality - for example sorting and searching events based on their data element values. api/events/query has support for this. Below are some examples

A query to return an event grid containing only selected data elements for a program stage

```
/api/33/events/query.json?orgUnit=DiszpKrYNg8&programStage=Zj7UnCAulEk
  &dataElement=qrur9Dvnyt5,fWIAEtYVEGk,K6uUAvq500H&order=lastUpdated:desc
  &pageSize=50&page=1&totalPages=true
```

A query to return an event grid containing all data elements of a program stage

```
/api/33/events/query.json?orgUnit=DiszpKrYNg8&programStage=Zj7UnCAulEk
  &includeAllDataElements=true
```

A query to filter events based on data element value

```
/api/33/events/query.json?orgUnit=DiszpKrYNg8&programStage=Zj7UnCAulEk
  &filter=qrur9Dvnyt5:GT:20:LT:50
```

In addition to the filtering, the above example also illustrates one thing: the fact that there are no data elements mentioned to be returned in the grid. When this happens, the system defaults back to return only those data elements marked "Display in report" under program stage configuration.

We can also extend the above query to return us a grid sorted (asc|desc) based on data element value

```
/api/33/events/query.json?orgUnit=DiszpKrYNg8&programStage=Zj7UnCAulEk
  &filter=qrur9Dvnyt5:GT:20:LT:50&order=qrur9Dvnyt5:desc
```

Event filters

To create, read, update and delete event filters you can interact with the `/api/eventFilters` resource.

```
/api/33/eventFilters
```

Create and update an event filter definition

For creating and updating an event filter in the system, you will be working with the *eventFilters* resource. *POST* is used to create and *PUT* method is used to update. The event filter definitions are used in the Tracker Capture app to display relevant predefined "Working lists" in the tracker user interface.

Request Payload

Request Property	Description	Example
name	Name of the filter.	"name":"My working list"
description	A description of the filter.	"description":"for listing all events assigned to me".
program	The uid of the program.	"program" : "a3kGcGDCuk6"
programStage	The uid of the program stage.	"programStage" : "a3kGcGDCuk6"
eventQueryCriteria	Object containing parameters for querying, sorting and filtering events.	"eventQueryCriteria": { "organisationUnit":"a3kGcGDCuk6", "status": "COMPLETED", "createdDate": { "from": "2014-05-01", "to": "2019-03-20" }, "dataElements": ["a3kGcGDCuk6:EQ:1", "a3kGcGDCuk6"], "filters": ["a3kGcGDCuk6:EQ:1"], "programStatus": "ACTIVE", "ouMode": "SELECTED", "assignedUserMode": "PROVIDED", "assignedUsers" : ["a3kGcGDCuk7", "a3kGcGDCuk8"], "followUp": false, "trackedEntityInstance": "a3kGcGDCuk6", "events": ["a3kGcGDCuk7", "a3kGcGDCuk8"], "fields": "eventDate,dueDate", "order": "dueDate:asc,createdDate:desc" }

Event Query Criteria definition

followUp	Used to filter events based on enrollment followUp flag. Possible values are true|false.	"followUp": true
organisationUnit	To specify the uid of the organisation unit	"organisationUnit": "a3kGcGDCuk7"
ouMode	To specify the OU selection mode. Possible values are SELECTED| CHILDREN| DESCENDANTS|ACCESSIBLE| CAPTURE|ALL	"ouMode": "SELECTED"
assignedUserMode	To specify the assigned user selection mode for events. Possible values are CURRENT| PROVIDED| NONE | ANY. See table below to understand what each value indicates. If PROVIDED (or null), non-empty assignedUsers in the payload will be considered.	"assignedUserMode": "PROVIDED"
assignedUsers	To specify a list of assigned users for events. To be used along with PROVIDED assignedUserMode above.	"assignedUsers": ["a3kGcGDCuk7", "a3kGcGDCuk8"]
displayOrderColumns	To specify the output ordering of columns	"displayOrderColumns": ["eventDate", "dueDate", "program"]
order	To specify ordering/sorting of fields and its directions in comma separated values. A single item in order is of the form "dataItem:direction".	"order"="a3kGcGDCuk6:desc,eventDate:asc"
dataFilters	To specify filters to be applied when listing events	"dataFilters"=[{ "dataItem": "abcDataElementUid", "le": "20", "ge": "10", "lt": "20", "gt": "10", "in": ["India", "Norway"], "like": "abc", "dateFilter": { "startDate": "2014-05-01", "endDate": "2019-03-20", "startBuffer": -5, "endBuffer": 5, "period": "LAST_WEEK", "type": "RELATIVE" } }]
status	Any valid EventStatus	"eventStatus": "COMPLETED"
events	To specify list of events	"events"=["a3kGcGDCuk6"]
completedDate	DateFilterPeriod object date filtering based on completed date.	"completedDate": { "startDate": "2014-05-01", "endDate": "2019-03-20", "startBuffer": -5, "endBuffer": 5, "period": "LAST_WEEK", "type": "RELATIVE" }

eventDate	DateFilterPeriod object date filtering based on event date.	"eventDate": { "startBuffer": -5, "endBuffer": 5, "type": "RELATIVE" }
dueDate	DateFilterPeriod object date filtering based on due date.	"dueDate": { "period": "LAST_WEEK", "type": "RELATIVE" }
lastUpdatedDate	DateFilterPeriod object date filtering based on last updated date.	"lastUpdatedDate": { "startDate": "2014-05-01", "endDate": "2019-03-20", "type": "ABSOLUTE" }

DateFilterPeriod object definition

type	Specify whether the date period type is ABSOLUTE | RELATIVE	"type" : "RELATIVE"
period	Specify if a relative system defined period is to be used. Applicable only when "type" is RELATIVE. (see Relative Periods for supported relative periods)	"period" : "THIS_WEEK"
startDate	Absolute start date. Applicable only when "type" is ABSOLUTE	"startDate":"2014-05-01"
endDate	Absolute end date. Applicable only when "type" is ABSOLUTE	"startDate":"2014-05-01"
startBuffer	Relative custom start date. Applicable only when "type" is RELATIVE	"startBuffer":-10
endBuffer	Relative custom end date. Applicable only when "type" is RELATIVE	"startDate":+10

The available assigned user selection modes are explained in the following table.

Assigned user selection modes (event assignment)

Mode	Description
CURRENT	Assigned to the current logged in user
PROVIDED	Assigned to the users provided in the "assignedUser" parameter
NONE	Assigned to no users.
ANY	Assigned to any users.

A sample payload that can be used to create/update an eventFilter is shown below.

```
{
    "program": "ur1Edk5Oe2n",
    "description": "Simple Filter for TB events",
    "name": "TB events",
    "eventQueryCriteria": {
        "organisationUnit": "DiszpKrYNg8",
        "eventStatus": "COMPLETED",
        "eventDate": {
```

```
            "startDate": "2014-05-01",
            "endDate": "2019-03-20",
            "startBuffer": -5,
            "endBuffer": 5,
            "period": "LAST_WEEK",
            "type": "RELATIVE"
        },
        "dataFilters": [
            {
                "dataItem": "abcDataElementUid",
                "le": "20",
                "ge": "10",
                "lt": "20",
                "gt": "10",
                "in": ["India", "Norway"],
                "like": "abc"
            },
            {
                "dataItem": "dateDataElementUid",
                "dateFilter": {
                    "startDate": "2014-05-01",
                    "endDate": "2019-03-20",
                    "type": "ABSOLUTE"
                }
            },
            {
                "dataItem": "anotherDateDataElementUid",
                "dateFilter": {
                    "startBuffer": -5,
                    "endBuffer": 5,
                    "type": "RELATIVE"
                }
            },
            {
                "dataItem": "yetAnotherDateDataElementUid",
                "dateFilter": {
                    "period": "LAST_WEEK",
                    "type": "RELATIVE"
                }
            }
        ],
        "programStatus": "ACTIVE"
    }
}
```

Retrieving and deleting event filters

A specific event filter can be retrieved by using the following api

```
GET /api/33/eventFilters/{uid}
```

All event filters can be retrieved by using the following api.

```
GET /api/33/eventFilters?fields=*
```

All event filters for a specific program can be retrieved by using the following api

```
GET /api/33/eventFilters?filter=program:eq:IpHINAT79UW
```

An event filter can be deleted by using the following api

```
DELETE /api/33/eventFilters/{uid}
```

Relationships

Relationships are links between two entities in tracker. These entities can be tracked entity instances, enrollments and events.

There are multiple endpoints that allow you to see, create, delete and update relationships. The most common is the /api/trackedEntityInstances endpoint, where you can include relationships in the payload to create, update or deleting them if you omit them - Similar to how you work with enrollments and events in the same endpoint. All the tracker endpoints, /api/trackedEntityInstances, /api/enrollments and /api/events also list their relationships if requested in the field filter.

The standard endpoint for relationships is, however, /api/relationships. This endpoint provides all the normal CRUD operations for relationships.

List all relationships require you to provide the UID of the trackedEntityInstance, Enrollment or event that you want to list all the relationships for:

```
GET /api/relationships?tei=ABCDEF12345
GET /api/relationships?enrollment=ABCDEF12345
GET /api/relationships?event=ABCDEF12345
```

This request will return a list of any relationship you have access to see that includes the trackedEntityInstance, enrollment or event you specified. Each relationship is represented with the following JSON:

```
{
    "relationshipType": "dDrh5UyCyvQ",
    "relationshipName": "Mother-Child",
    "relationship": "t0HIBrc65Rm",
    "bidirectional": false,
    "from": {
        "trackedEntityInstance": {
            "trackedEntityInstance": "vOxUH373fy5"
        }
    },
    "to": {
        "trackedEntityInstance": {
            "trackedEntityInstance": "pybd813kIWx"
        }
    },
    "created": "2019-04-26T09:30:56.267",
    "lastUpdated": "2019-04-26T09:30:56.267"
}
```

You can also view specified relationships using the following endpoint:

```
GET /api/relationships/<id>
```

To create or update a relationship, you can use the following endpoints:

```
POST /api/relationships
PUT /api/relationships
```

And use the following payload structure:

```
{
    "relationshipType": "dDrh5UyCyvQ",
    "from": {
        "trackedEntityInstance": {
            "trackedEntityInstance": "vOxUH373fy5"
        }
    },
    "to": {
        "trackedEntityInstance": {
            "trackedEntityInstance": "pybd813kIWx"
        }
    }
}
```

To delete a relationship, you can use this endpoint:

```
  DELETE /api/relationships/<id>
```

In our example payloads, we use a relationship between trackedEntityInstances. Because of this, the "from" and "to" properties of our payloads include "trackedEntityInstance" objects. If your relationship includes other entities, you can use the following properties:

```
{
    "enrollment": {
        "enrollment": "<id>"
    }
}
```

```
{
    "event": {
        "event": "<id>"
    }
}
```

Update strategies

Two update strategies for all 3 tracker endpoints are supported: enrollment and event creation. This is useful when you have generated an identifier on the client side and are not sure if it was created or not on the server.

Available tracker strategies

Parameter	Description
CREATE	Create only, this is the default behavior.
CREATE_AND_UPDATE	Try and match the ID, if it exist then update, if not create.

To change the parameter, please use the strategy parameter:

```
POST /api/33/trackedEntityInstances?strategy=CREATE_AND_UPDATE
```

Tracker bulk deletion

Bulk deletion of tracker objects work in a similar fashion to adding and updating tracker objects, the only difference is that the `importStrategy` is *DELETE*.

Example: Bulk deletion of tracked entity instances:

```
{
    "trackedEntityInstances": [
        {
            "trackedEntityInstance": "ID1"
        },
        {
            "trackedEntityInstance": "ID2"
        },
        {
            "trackedEntityInstance": "ID3"
        }
    ]
}
```

```
curl -X POST -d @data.json -H "Content-Type: application/json"
  "http://server/api/33/trackedEntityInstances?strategy=DELETE"
```

Example: Bulk deletion of enrollments:

```
{
    "enrollments": [
        {
            "enrollment": "ID1"
        },
        {
            "enrollment": "ID2"
        },
        {
            "enrollment": "ID3"
        }
    ]
}
```

```
curl -X POST -d @data.json -H "Content-Type: application/json"
  "http://server/api/33/enrollments?strategy=DELETE"
```

Example: Bulk deletion of events:

```
{
    "events": [
        {
            "event": "ID1"
        },
        {
            "event": "ID2"
        },
        {
            "event": "ID3"
        }
    ]
}
```

```
curl -X POST -d @data.json -H "Content-Type: application/json"
  "http://server/api/33/events?strategy=DELETE"
```

Identifier reuse and item deletion via POST and PUT methods

Tracker endpoints */trackedEntityInstances*, */enrollments*, */events* support CRUD operations. The system keeps track of used identifiers. Therefore, an item which has been created and then deleted (e.g. events, enrollments) cannot be created or updated again. If attempting to delete an already deleted item, the system returns a success response as deletion of an already deleted item implies no change.

The system does not allow to delete an item via an update (*PUT*) or create (*POST*) method. Therefore, an attribute *deleted* is ignored in both *PUT* and *POST* methods, and in *POST* method it is by default set to *false*.

Import parameters

The import process can be customized using a set of import parameters:

Import parameters

Parameter	Values (default first)	Description
dataElementIdScheme	id | name | code | attribute:ID	Property of the data element object to use to map the data values.
orgUnitIdScheme	id | name | code | attribute:ID	Property of the org unit object to use to map the data values.
idScheme	id | name | code| attribute:ID	Property of all objects including data elements, org units and category option combos, to use to map the data values.
dryRun	false | true	Whether to save changes on the server or just return the import summary.
strategy	CREATE | UPDATE | CREATE_AND_UPDATE | DELETE	Save objects of all, new or update import status on the server.

Parameter	Values (default first)	Description
skipNotifications	true | false	Indicates whether to send notifications for completed events.
skipFirst	true | false	Relevant for CSV import only. Indicates whether CSV file contains a header row which should be skipped.
importReportMode	FULL, ERRORS, DEBUG	Sets the `ImportReport` mode, controls how much is reported back after the import is done. `ERRORS` only includes *Object Reports* for object which has errors. `FULL` returns an *Object Report* for all objects imported, and `DEBUG` returns the same plus a name for the object (if available).

CSV Import / Export

In addition to XML and JSON for event import/export, in DHIS2.17 we introduced support for the CSV format. Support for this format builds on what was described in the last section, so here we will only write about what the CSV specific parts are.

To use the CSV format you must either use the `/api/events.csv` endpoint, or add *content-type: text/csv* for import, and *accept: text/csv* for export when using the `/api/events` endpoint.

The order of column in the CSV which are used for both export and import is as follows:

CSV column

Index	Key	Type	Description
1	event	identifier	Identifier of event
2	status	enum	Status of event, can be ACTIVE | COMPLETED | VISITED | SCHEDULED | OVERDUE | SKIPPED
3	program	identifier	Identifier of program
4	programStage	identifier	Identifier of program stage
5	enrollment	identifier	Identifier of enrollment (program instance)
6	orgUnit	identifier	Identifier of organisation unit
7	eventDate	date	Event date
8	dueDate	date	Due Date
9	latitude	double	Latitude where event happened

Index	Key	Type	Description
10	longitude	double	Longitude where event happened
11	dataElement	identifier	Identifier of data element
12	value	string	Value / measure of event
13	storedBy	string	Event was stored by (defaults to current user)
14	providedElsewhere	boolean	Was this value collected somewhere else
14	completedDate	date	Completed date of event
14	completedBy	string	Username of user who completed event

Example of 2 events with 2 different data value each:

```
EJNxP3WreNP,COMPLETED,<pid>,<psid>,<enrollment-id>,<ou>,2016-01-01,2016-01-01,,,<de>,1,,
EJNxP3WreNP,COMPLETED,<pid>,<psid>,<enrollment-id>,<ou>,2016-01-01,2016-01-01,,,<de>,2,,
qPEdI1xn7k0,COMPLETED,<pid>,<psid>,<enrollment-id>,<ou>,2016-01-01,2016-01-01,,,<de>,3,,
qPEdI1xn7k0,COMPLETED,<pid>,<psid>,<enrollment-id>,<ou>,2016-01-01,2016-01-01,,,<de>,4,,
```

Import strategy: SYNC

The import strategy SYNC should be used only by internal synchronization task and not for regular import. The SYNC strategy allows all 3 operations: CREATE, UPDATE, DELETE to be present in the payload at the same time.

Tracker Ownership Management

A new concept called Tracker Ownership is introduced from 2.30. There will now be one owner organisation unit for a tracked entity instance in the context of a program. Programs that are configured with an access level of *PROTECTED* or *CLOSED* will adhere to the ownership privileges. Only those users belonging to the owning org unit for a tracked entity-program combination will be able to access the data related to that program for that tracked entity.

Tracker Ownership Override : Break the Glass

It is possible to temporarily override this ownership privilege for a program that is configured with an access level of *PROTECTED*. Any user will be able to temporarily gain access to the program related data, if the user specifies a reason for accessing the tracked entity-program data. This act of temporarily gaining access is termed as *breaking the glass*. Currently, the temporary access is granted for 3 hours. DHIS2 audits breaking the glass along with the reason specified by the user. It is not possible to gain temporary access to a program that has been configured with an access level of *CLOSED*. To break the glass for a tracked entity program combination, you can issue a POST request as shown:

```
/api/33/tracker/ownership/override?trackedEntityInstance=DiszpKrYNg8
  &program=eBAyeGv0exc&reason=patient+showed+up+for+emergency+care
```

Tracker Ownership Transfer

It is possible to transfer the ownership of a tracked entity-program from one org unit to another. This will be useful in case of patient referrals or migrations. Only an owner (or users who have broken the glass) can transfer the ownership. To transfer ownership of a tracked entity-program to another organisation unit, you can issue a PUT request as shown:

```
/api/33/tracker/ownership/transfer?trackedEntityInstance=DiszpKrYNg8
  &program=eBAyeGv0exc&ou=EJNxP3WreNP
```

Potential Duplicates

Potential duplicates are records we work with in the data deduplication feature. Due to the nature of the deduplication feature, this API endpoint is somewhat restricted.

A potential duplicate represents a single or pair of records which are suspected to be a duplicate.

The payload of a potential duplicate looks like this:

```
{
    "teiA": "<id>",
    "teiB": "<id>",
    "status": "OPEN|INVALID|MERGED"
}
```

You can retrieve a list of potential duplicates using the following endpoint:

```
GET /api/potentialDuplicates
```

Additionally you can inspect individual records:

```
GET /api/potentialDuplicates/<id>
```

To create a new potential duplicate, you can use this endpoint:

```
POST /api/potentialDuplicates
```

The payload you provide needs at least *teiA* to be a valid tracked entity instance; *teiB* is optional. If *teiB* is set, it also needs to point to an existing tracked entity instance.

```
{
    "teiA": "<id>",
    "teiB": "<id>"
}
```

You can mark a potential duplicate as *invalid* to tell the system that the potential duplicate has been investigated and deemed to be not a duplicate. To do so you can use the following endpoint:

```
PUT /api/potentialDuplicates/<id>/invalidation
```

To hard delete a potential duplicate:

```
DELETE /api/potentialDuplicates/<id>
```

Email

The Web API features a resource for sending emails. For emails to be sent it is required that the SMTP configuration has been properly set up and that a system notification email address for the DHIS2 instance has been defined. You can set SMTP settings from the email settings screen and system notification email address from the general settings screen in DHIS2.

```
/api/33/email
```

System notification

The *notification* resource lets you send system email notifications with a given subject and text in JSON or XML. The email will be sent to the notification email address as defined in the DHIS2 general system settings:

```
{
    "subject": "Integrity check summary",
    "text": "All checks ran successfully"
}
```

You can send a system email notification by posting to the notification resource like this:

```
curl -d @email.json "localhost/api/33/email/notification" -X POST
  -H "Content-Type:application/json" -u admin:district
```

Outbound emails

You can also send a general email notification by posting to the notification resource as mentioned below. `F_SEND_EMAIL` or `ALL` authority has to be in the system to make use of this api. Subject parameter is optional. "DHIS 2" string will be sent as default subject if it is not provided in url. Url should be encoded in order to use this API.

```
curl "localhost/api/33/email/notification?
recipients=xyz%40abc.com&message=sample%20email&subject=Test%20Email"
  -X POST -u admin:district
```

Test message

To test whether the SMTP setup is correct by sending a test email to yourself you can interact with the *test* resource. To send test emails it is required that your DHIS2 user account has a valid email address associated with it. You can send a test email like this:

```
curl "localhost/api/33/email/test" -X POST -H "Content-Type:application/json" -u admin:district
```

Sharing

The sharing solution allows you to share most objects in the system with specific user groups and to define whether objects should be publicly accessible or private. To get and set sharing status for objects you can interact with the *sharing* resource.

```
/api/33/sharing
```

Get sharing status

To request the sharing status for an object use a GET request to:

```
/api/33/sharing?type=dataElement&id=fbfJHSPpUQD
```

The response looks like the below.

```
{
    "meta": {
        "allowPublicAccess": true,
        "allowExternalAccess": false
    },
    "object": {
        "id": "fbfJHSPpUQD",
        "name": "ANC 1st visit",
        "publicAccess": "rw------",
        "externalAccess": false,
        "user": {},
        "userGroupAccesses": [
            {
                "id": "hj0nnsVsPLU",
                "access": "rw------"
            },
            {
                "id": "qMjBflJMOfB",
                "access": "r-------"
            }
        ]
    }
}
```

Set sharing status

You can define the sharing status for an object using the same URL with a POST request, where the payload in JSON format looks like this:

```
{
    "object": {
        "publicAccess": "rw------",
        "externalAccess": false,
        "user": {},
        "userGroupAccesses": [
            {
                "id": "hj0nnsVsPLU",
                "access": "rw------"
            },
            {
                "id": "qMjBflJMOfB",
                "access": "r-------"
            }
        ]
    }
}
```

In this example, the payload defines the object to have read-write public access, no external access (without login), read-write access to one user group and read-only access to another user group. You can submit this to the sharing resource using curl:

```
curl -d @sharing.json "localhost/api/33/sharing?type=dataElement&id=fbfJHSPpUQD"
  -H "Content-Type:application/json" -u admin:district
```

Scheduling

DHIS2 allows for scheduling of jobs of various types. Each type of job has different properties for configuration, giving you finer control over how jobs are run. In addition, you can configure the same job to run with different configurations and at different intervals if required.

Main properties

Property	Description	Type
name	Name of the job.	String
cronExpression	The cron expression which defines the interval for when the job should run.	String (Cron expression)
jobType	The job type represent which task is run. In the next table, you can get an overview of existing job types. Each job type can have a specific set of parameters for job configuration.	String (Enum)
jobParameters	Job parameters, if applicable for job type.	(See list of job types)
enabled	A job can be added to the system without it being scheduled by setting `enabled` to false in the JSON payload. Use this if you want to temporarily stop scheduling for a job, or if a job configuration is not complete yet.	Boolean

Available job types

Job type	Parameters	Param(Type:Default)
DATA_INTEGRITY	NONE	
ANALYTICS_TABLE	• lastYears: Number of years back to include • skipTableTypes: Skip generation of tables Possible values: DATA_VALUE, COMPLETENESS, COMPLETENESS_TARGET, ORG_UNIT_TARGET, EVENT, ENROLLMENT, VALIDATION_RESULT • skipResourceTables: Skip generation of resource tables	• lastYears (int:0) • skipTableTypes (Array of String (Enum):None) • skipResourceTables (Boolean)
CONTINUOUS_ANALYTICS_TABLE	• fullUpdateHourOfDay: Hour of day for full update of analytics tables (0-23) • lastYears: Number of years back to include • skipTableTypes: Skip generation of tables Possible values: DATA_VALUE, COMPLETENESS, COMPLETENESS_TARGET, ORG_UNIT_TARGET, EVENT, ENROLLMENT, VALIDATION_RESULT • skipResourceTables: Skip generation of resource tables	• lastYears (int:0) • skipTableTypes (Array of String (Enum):None) • skipResourceTables (Boolean)
DATA_SYNC	NONE	
META_DATA_SYNC	NONE	
SEND_SCHEDULED_MESSAGE	NONE	
PROGRAM_NOTIFICATIONS	NONE	

Job type	Parameters	Param(Type:Default)
MONITORING (Validation rule analysis)	• relativeStart: A number related to date of execution which resembles the start of the period to monitor • relativeEnd: A number related to date of execution which resembles the end of the period to monitor • validationRuleGroups: Validation rule groups(UIDs) to include in job • sendNotification: Set "true" if job should send notifications based on validation rule groups • persistsResults: Set "true" if job should persist validation results	• relativeStart (int:0) • relativeEnd (int:0) • validationRuleGroups (Array of String (UIDs):None) • sendNotification (Boolean:false) • persistsResults (Boolean:false)
PUSH_ANALYSIS	• pushAnalysis: The uid of the push analysis you want to run	• pushAnalysis (String:None)
PREDICTOR	• relativeStart: A number related to date of execution which resembles the start of the period to monitor • relativeEnd: A number related to date of execution which resembles the start of the period to monitor • predictors: Predictors(UIDs) to include in job	• relativeStart (int:0) • relativeEnd (int:0) • predictors (Array of String (UIDs):None)

Get available job types

To get a list of all available job types you can use the following endpoint:

```
GET /api/jobConfigurations/jobTypes
```

The response contains information about each job type including name, job type, key, scheduling type and available parameters. The scheduling type can either be `CRON`, meaning jobs can be scheduled using a cron expression with the `cronExpression` field, or `FIXED_DELAY`, meaning jobs can be scheduled to run with a fixed delay in between with the `delay` field. The field delay is given in seconds.

A response will look similar to this:

```
{
    "jobTypes": [
        {
            "name": "Data integrity",
            "jobType": "DATA_INTEGRITY",
            "key": "dataIntegrityJob",
            "schedulingType": "CRON"
        },
        {
            "name": "Resource table",
            "jobType": "RESOURCE_TABLE",
            "key": "resourceTableJob",
            "schedulingType": "CRON"
        },
        {
            "name": "Continuous analytics table",
            "jobType": "CONTINUOUS_ANALYTICS_TABLE",
            "key": "continuousAnalyticsTableJob",
            "schedulingType": "FIXED_DELAY"
        }
    ]
}
```

Create job

To configure jobs you can do a POST request to the following resource:

```
/api/jobConfigurations
```

A job without parameters in JSON format looks like this :

```
{
    "name": "",
    "jobType": "JOBTYPE",
    "cronExpression": "0 * * ? * *"
}
```

An example of an analytics table job with parameters in JSON format:

```
{
    "name": "Analytics tables last two years",
    "jobType": "ANALYTICS_TABLE",
    "cronExpression": "0 * * ? * *",
    "jobParameters": {
        "lastYears": "2",
        "skipTableTypes": [],
        "skipResourceTables": false
```

```
    }
}
```

As example of a push analysis job with parameters in JSON format:

```
{
    "name": "Push anlysis charts",
    "jobType": "PUSH_ANALYSIS",
    "cronExpression": "0 * * ? * *",
    "jobParameters": {
        "pushAnalysis": ["jtcMAKhWwnc"]
    }
}
```

An example of a job with scheduling type `FIXED_DELAY` and 120 seconds delay:

```
{
    "name": "Continuous analytics table",
    "jobType": "CONTINUOUS_ANALYTICS_TABLE",
    "delay": "120",
    "jobParameters": {
        "fullUpdateHourOfDay": 4
    }
}
```

Get jobs

List all job configurations:

```
GET /api/jobConfigurations
```

Retrieve a job:

```
GET /api/jobConfigurations/{id}
```

The response payload looks like this:

```
{
    "lastUpdated": "2018-02-22T15:15:34.067",
    "id": "KBcP6Qw37gT",
    "href": "http://localhost:8080/api/jobConfigurations/KBcP6Qw37gT",
    "created": "2018-02-22T15:15:34.067",
    "name": "analytics last two years",
    "jobStatus": "SCHEDULED",
    "displayName": "analytics last two years",
    "enabled": true,
    "externalAccess": false,
    "jobType": "ANALYTICS_TABLE",
    "nextExecutionTime": "2018-02-26T03:00:00.000",
    "cronExpression": "0 0 3 ? * MON",
    "jobParameters": {
        "lastYears": 2,
        "skipTableTypes": [],
        "skipResourceTables": false
```

```
    },
    "favorite": false,
    "configurable": true,
    "access": {
        "read": true,
        "update": true,
        "externalize": true,
        "delete": true,
        "write": true,
        "manage": true
    },
    "lastUpdatedBy": {
        "id": "GOLswS44mh8"
    },
    "favorites": [],
    "translations": [],
    "userGroupAccesses": [],
    "attributeValues": [],
    "userAccesses": []
}
```

Update job

Update a job with parameters using the following endpoint and JSON payload format:

```
PUT /api/jobConfiguration/{id}
```

```
{
    "name": "analytics last two years",
    "enabled": true,
    "cronExpression": "0 0 3 ? * MON",
    "jobType": "ANALYTICS_TABLE",
    "jobParameters": {
        "lastYears": "3",
        "skipTableTypes": [],
        "skipResourceTables": false
    }
}
```

Delete job

Delete a job using:

```
DELETE /api/jobConfiguration/{id}
```

Note that some jobs with custom configuration parameters may not be added if the required system settings are not configured. An example of this is data synchronization, which requires remote server configuration.

Schema

A resource which can be used to introspect all available DXF 2 objects can be found on `/api/schemas`. For specific resources you can have a look at `/api/schemas/<type>`.

To get all available schemas in XML:

```
GET /api/schemas.xml
```

To get all available schemas in JSON:

```
GET /api/schemas.json
```

To get JSON schema for a specific class:

```
GET /api/schemas/dataElement.json
```

UI customization

To customize the UI of the DHIS2 application you can insert custom JavaScript and CSS styles through the *files* resource.

```
POST GET DELETE /api/33/files/script
POST GET DELETE /api/33/files/style
```

The JavaScript and CSS content inserted through this resource will be loaded by the DHIS2 web application. This can be particularly useful in certain situations:

- Overriding the CSS styles of the DHIS2 application, such as the login page or main page.
- Defining JavaScript functions which are common to several custom data entry forms and HTML-based reports.
- Including CSS styles which are used in custom data entry forms and HTML-based reports.

Javascript

To insert Javascript from a file called *script.js* you can interact with the *files/script* resource with a POST request:

```
curl --data-binary @script.js "localhost/api/33/files/script"
  -H "Content-Type:application/javascript" -u admin:district
```

Note that we use the `--data-binary` option to preserve formatting of the file content. You can fetch the JavaScript content with a GET request:

```
/api/33/files/script
```

To remove the JavaScript content you can use a DELETE request.

CSS

To insert CSS from a file called *style.css* you can interact with the *files/style* resource with a POST-request:

```
curl --data-binary @style.css "localhost/api/33/files/style"
  -H "Content-Type:text/css" -u admin:district
```

You can fetch the CSS content with a GET-request:

```
/api/33/files/style
```

To remove the JavaScript content you can use a DELETE request.

Synchronization

This section covers pull and push of data and metadata.

Data value push

To initiate a data value push to a remote server one must first configure the URL and credentials for the relevant server from System settings > Synchronization, then make a POST request to the following resource:

```
/api/33/synchronization/dataPush
```

Metadata pull

To initiate a metadata pull from a remote JSON document you can make a POST request with a *url* as request payload to the following resource:

```
/api/33/synchronization/metadataPull
```

Availability check

To check the availability of the remote data server and verify user credentials you can make a GET request to the following resource:

```
/api/33/synchronization/availability
```

Apps

The `/api/apps` endpoint can be used for installing, deleting and listing apps. The app key is based on the app name, but with all non-alphanumerical characters removed, and spaces replaced with a dash. *My app!* will return the key *My-app*.

> **Note**
>
> Previous to 2.28, the app key was derived from the name of the ZIP archive, excluding the file extension. URLs using the old format should still return the correct app in the api.

```
/api/33/apps
```

Get apps

> **Note**
>
> Previous to 2.28 the app property folderName referred to the actual path of the installed app. With the ability to store apps on cloud services, folderName's purpose changed, and will now refer to the app key.

You can read the keys for apps by listing all apps from the apps resource and look for the *key* property. To list all installed apps in JSON:

```
curl -u user:pass -H "Accept: application/json" "http://server.com/api/33/apps"
```

You can also simply point your web browser to the resource URL:

```
http://server.com/api/33/apps
```

The apps list can also be filtered by app type and by name, by appending one or more *filter* parameters to the URL:

```
http://server.com/api/33/apps?filter=appType:eq:DASHBOARD_APP&filter=name:ilike:youtube
```

App names support the *eq* and *ilike* filter operators, while *appType* supports *eq* only.

Install an app

To install an app, the following command can be issued:

```
curl -X POST -u user:pass -F file=@app.zip "http://server.com/api/33/apps"
```

Delete an app

To delete an app, you can issue the following command:

```
curl -X DELETE -u user:pass "http://server.com/api/33/apps/<app-key>"
```

Reload apps

To force a reload of currently installed apps, you can issue the following command. This is useful if you added a file manually directly to the file system, instead of uploading through the DHIS2 user interface.

```
curl -X PUT -u user:pass "http://server.com/api/33/apps"
```

Share apps between instances

If the DHIS2 instance has been configured to use cloud storage, apps will now be installed and stored on the cloud service. This will enable multiple instances share the same versions on installed apps, instead of installing the same apps on each individual instance.

> **Note**
>
> Previous to 2.28, installed apps would only be stored on the instance's local filesystem. Apps installed before 2.28 will still be available on the instance it was installed, but it will not be shared with other instances, as it's still located on the instances local filesystem.

App store

The Web API exposes the content of the DHIS2 App Store as a JSON representation which can found at the `/api/appStore` resource.

```
/api/33/appStore
```

Get apps

You can retrieve apps with a GET request:

```
GET /api/33/appStore
```

A sample JSON response is described below.


```
{
  [
    {
      "name": "Tabular Tracker Capture",
      "description": "Tabular Tracker Capture is an app that makes you more effective.",
      "sourceUrl": "https://github.com/dhis2/App-repository",
      "appType": "DASHBOARD_WIDGET",
      "status": "PENDING",
      "id": "NSD06BVoV21",
      "developer": {
        "name": "DHIS",
        "organisation": "Uio",
        "address": "Oslo",
        "email": "dhis@abc.com",
      },
      "versions": [
        {
          "id": "upAPqrVgwK6",
          "version": "1.2",
          "minDhisVersion": "2.17",
          "maxDhisVersion": "2.20",
          "downloadUrl": "https://dhis2.org/download/appstore/tabular-capture-12.zip",
          "demoUrl": "http://play.dhis2.org/demo"
        }
      ],
      "images": [
        {
          "id": "upAPqrVgwK6",
```

```
          "logo": "true",
          "imageUrl": "https://dhis2.org/download/appstore/tabular-capture-12.png",
          "description": "added feature snapshot",
          "caption": "dialog",
        }
      ]
    }
  ]
}
```

Install apps

You can install apps on your instance of DHIS2 assuming you have the appropriate permissions. An app is referred to using the `id` property of the relevant version of the app. An app is installed with a POST request with the version id to the following resource:

```
POST /api/33/appStore/{app-version-id}
```

Data store

Using the *dataStore* resource, developers can store arbitrary data for their apps. Access to a datastore's key is based on its sharing settings. By default all keys created are publicly accessible (read and write). Additionally, access to a datastore's namespace is limited to the user's access to the corresponding app, if the app has reserved the namespace. For example a user with access to the "sampleApp" application will also be able to use the sampleApp namespace in the datastore. If a namespace is not reserved, no specific access is required to use it.

```
/api/33/dataStore
```

Data store structure

Data store entries consist of a namespace, key and value. The combination of namespace and key is unique. The value data type is JSON.

Data store structure

Item	Description	Data type
Namespace	Namespace for organization of entries.	String
Key	Key for identification of values.	String
Value	Value holding the information for the entry.	JSON
Encrypted	Indicates whether the value of the given key should be encrypted	Boolean

Get keys and namespaces

For a list of all existing namespaces:

```
GET /api/33/dataStore
```

Example curl request for listing:

```
curl "play.dhis2.org/demo/api/33/dataStore" -u admin:district
```

Example response:

```
["foo", "bar"]
```

For a list of all keys in a namespace:

```
GET /api/33/dataStore/<namespace>
```

Example curl request for listing:

```
curl "play.dhis2.org/demo/api/33/dataStore/foo" -u admin:district
```

Example response:

```
["key_1", "key_2"]
```

To retrieve a value for an existing key from a namespace:

```
GET /api/33/dataStore/<namespace>/<key>
```

Example curl request for retrieval:

```
curl "play.dhis2.org/demo/api/33/dataStore/foo/key_1"-u admin:district
```

Example response:

```
{
    "foo": "bar"
}
```

To retrieve meta-data for an existing key from a namespace:

```
GET /api/33/dataStore/<namespace>/<key>/metaData
```

Example curl request for retrieval:

```
curl "play.dhis2.org/demo/api/33/dataStore/foo/key_1/metaData" -u admin:district
```

Example response:

```
{
  "created": "...",
  "user": {...},
  "namespace": "foo",
  "key": "key_1"
}
```

Create values

To create a new key and value for a namespace:

```
POST /api/33/dataStore/<namespace>/<key>
```

Example curl request for create, assuming a valid JSON payload:

```
curl "https://play.dhis2.org/demo/api/33/dataStore/foo/key_1" -X POST
  -H "Content-Type: application/json" -d "{\"foo\":\"bar\"}" -u admin:district
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 201,
    "status": "OK",
    "message": "Key 'key_1' created."
}
```

If you require the data you store to be encrypted (for example user credentials or similar) you can append a query to the url like this:

```
GET /api/33/dataStore/<namespace>/<key>?encrypt=true
```

Update values

To update a key that exists in a namespace:

```
PUT /api/33/dataStore/<namespace>/<key>
```

Example curl request for update, assuming valid JSON payload:

```
curl "https://play.dhis2.org/demo/api/33/dataStore/foo/key_1" -X PUT -d "[1, 2, 3]"
  -H "Content-Type: application/json" -u admin:district
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
```

```
    "message": "Key 'key_1' updated."
}
```

Delete keys

To delete an existing key from a namespace:

```
DELETE /api/33/dataStore/<namespace>/<key>
```

Example curl request for delete:

```
curl "play.dhis2.org/demo/api/33/dataStore/foo/key_1" -X DELETE -u admin:district
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Key 'key_1' deleted from namespace 'foo'."
}
```

To delete all keys in a namespace:

```
DELETE /api/33/dataStore/<namespace>
```

Example curl request for delete:

```
curl "play.dhis2.org/demo/api/33/dataStore/foo" -X DELETE -u admin:district
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Namespace 'foo' deleted."
}
```

Sharing datastore keys

Sharing of datastore keys follows the same principle as for other metadata sharing (see Sharing).

To get sharing settings for a specific datastore key:

```
GET /api/33/sharing?type=dataStore&id=<uid>
```

To modify sharing settings for a specific datastore key:

```
POST /api/33/sharing?type=dataStore&id=<uid>
```

with the following request:

```
{
    "object": {
        "publicAccess": "rw------",
        "externalAccess": false,
        "user": {},
        "userAccesses": [],
        "userGroupAccesses": [
            {
                "id": "hj0nnsVsPLU",
                "access": "rw------"
            },
            {
                "id": "qMjBflJMOfB",
                "access": "r-------"
            }
        ]
    }
}
```

User data store

In addition to the *dataStore* which is shared between all users of the system, a user-based data store is also available. Data stored to the *userDataStore* is associated with individual users, so that each user can have different data on the same namespace and key combination. All calls against the *userDataStore* will be associated with the logged in user. This means one can only see, change, remove and add values associated with the currently logged in user.

```
/api/33/userDataStore
```

User data store structure

userDataStore consists of a user, a namespace, keys and associated values. The combination of user, namespace and key is unique.

User data store structure

Item	Description	Data Type
User	The user this data is associated with	String
Namespace	The namespace the key belongs to	String
Key	The key a value is stored on	String
Value	The value stored	JSON
Encrypted	Indicates whether the value should be encrypted	Boolean

Get namespaces

Returns an array of all existing namespaces

```
GET /api/33/userDataStore
```

Example request:

```
curl -H "Content-Type: application/json" -u admin:district "play.dhis2.org/api/33/userDataStore"
```

```
["foo", "bar"]
```

Get keys

Returns an array of all existing keys in a given namespace

```
GET /api/userDataStore/<namespace>
```

Example request:

```
curl -H "Content-Type: application/json" -u admin:district "play.dhis2.org/api/33/userDataStore/
foo"
```

```
["key_1", "key_2"]
```

Get values

Returns the value for a given namespace and key

```
GET /api/33/userDataStore/<namespace>/<key>
```

Example request:

```
curl -H "Content-Type: application/json" -u admin:district "play.dhis2.org/api/33/userDataStore/
foo/bar"
```

```
{
    "some": "value"
}
```

Create value

Adds a new value to a given key in a given namespace.

```
POST /api/33/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X POST -H "Content-Type: application/json" -u admin:district -d "['some value']"
  "play.dhis2.org/api/33/userDataStore/foo/bar"
```

```
{
    "httpStatus": "Created",
    "httpStatusCode": 201,
    "status": "OK",
    "message": "Key 'bar' in namespace 'foo' created."
}
```

If you require the value to be encrypted (For example user credentials and such) you can append a query to the url like this:

```
GET /api/33/userDataStore/<namespace>/<key>?encrypt=true
```

Update values

Updates an existing value

```
PUT /api/33/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X PUT -H "Content-Type: application/json" -u admin:district -d "['new value']"
  "play.dhis2.org/api/33/userDataStore/foo/bar"
```

```
{
    "httpStatus": "Created",
    "httpStatusCode": 201,
    "status": "OK",
    "message": "Key 'bar' in namespace 'foo' updated."
}
```

Delete key

Delete a key

```
DELETE /api/33/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X DELETE -u admin:district "play.dhis2.org/api/33/userDataStore/foo/bar"
```

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
```

```
    "message": "Key 'bar' deleted from the namespace 'foo."
}
```

Delete namespace

Delete all keys in the given namespace

```
DELETE /api/33/userDataStore/<namespace>
```

Example request:

```
curl -X DELETE -u admin:district "play.dhis2.org/api/33/userDataStore/foo"
```

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "All keys from namespace 'foo' deleted."
}
```

Predictors

A predictor allows you to generate data values based on an expression. This can be used for example to generate targets, thresholds, or estimated values.

To retrieve predictors you can make a GET request to the predictors resource like this:

```
/api/predictors
```

Creating a predictor

You can create a predictor with a POST request to the predictors resource:

```
POST /api/predictors
```

A sample payload looks like this:

```
{
    "id": "AG10KUJCrRk",
    "name": "Malaria Outbreak Threshold Predictor",
    "shortName": "Malaria Outbreak Predictor",
    "description": "Computes the threshold for potential malaria outbreaks based on the mean plus
1.5x the std dev",
    "output": {
        "id": "nXJJZNVAy0Y"
    },
    "generator": {
        "expression": "AVG(#{r6nrJANOqMw})+1.5*STDDEV(#{r6nrJANOqMw})",
        "description": "Maximum normal malaria case count",
        "missingValueStrategy": "NEVER_SKIP",
        "slidingWindow": false
    },
```

```
    "periodType": "Monthly",
    "sequentialSampleCount": 4,
    "sequentialSkipCount": 1,
    "annualSampleCount": 3,
    "organisationUnitLevels": [4]
}
```

The output element refers to the identifier of the data element for which to saved predicted data values. The generator element refers to the expression to use when calculating the predicted values.

Predictor expressions

A predictor always has a generator expression that describes how the predicted value is calculated. A predictor may also have a skip test expression returning a boolean value. When the skip test expression is present, it is evaluated in each of the sampled periods to tell whether values from that period should be skipped.

The following variables may be used in either a generator expression or a skip test expression:

Variable	Object	Description
#{\}	Aggregate data element	Refers to the total value of an aggregate data element across all category option combinations.
#{\.\	Data element operand	Refers to a combination of an aggregate data element and a category option combination.
D{\.\}	Program data element	Refers to the value of a tracker data element within a program.
A{\.\}	Program tracked entity attribute	Refers to the value of a tracked entity attribute within a program.
I{\}	Program indicator	Refers to the value of a program indicator.
R{\.\}	Reporting rate	Refers to a reporting rate metric. The metric can be REPORTING_RATE, REPORTING_RATE_ON_TIME, ACTUAL_REPORTS, ACTUAL_REPORTS_ON_TIME, EXPECTED_REPORTS.
C{\}	Constant	Refers to a constant value.
OUG{\}	Organisation unit group	Refers to the count of organisation units within an organisation unit group.
[days]	Number of days	The number of days in the current period.

Generating predicted values

To run all predictors (generating predicted values) you can make a POST request to the run resource:

```
POST /api/predictors/run
```

To run a single predictor you can make a POST request to the run resource for a predictor:

```
POST /api/predictors/AG10KUJCrRk/run
```

Min-max data elements

The min-max data elements resource allows you to set minimum and maximum value ranges for data elements. It is unique by the combination of organisation unit, data element and category option combo.

```
/api/minMaxDataElements
```

Min-max data element data structure

Item	Description	Data type
source	Organisation unit identifier	String
dataElement	Data element identifier	String
optionCombo	Data element category option combo identifier	String
min	Minimum value	Integer
max	Maximum value	Integer
generated	Indicates whether this object is generated by the system (and not set manually).	Boolean

You can retrieve a list of all min-max data elements from the following resource:

```
GET /api/minMaxDataElements.json
```

You can filter the response like this:

```
GET /api/minMaxDataElements.json?filter=dataElement.id:eq:UOlfIjgN8X6

GET /api/minMaxDataElements.json?filter=dataElement.id:in:[UOlfIjgN8X6,xc8gmAKfO95]
```

The filter parameter for min-max data elements supports two operators: eq and in. You can also use the `fields` query parameter.

```
GET /api/minMaxDataElements.json?fields=:all,dataElement[id,name]
```

Add/update min-max data element

To add a new min-max data element, use POST request to:

```
POST /api/minMaxDataElements.json
```

The JSON content format looks like this:

```
{
    "min": 1,
    "generated": false,
    "max": 100,
    "dataElement": {
        "id": "UOlfIjgN8X6"
    },
    "source": {
        "id": "DiszpKrYNg8"
    },
    "optionCombo": {
        "id": "psbwp3CQEhs"
    }
}
```

If the combination of data element, organisation unit and category option combo exists, the min-max value will be updated.

Delete min-max data element

To delete a min-max data element, send a request with DELETE method:

```
DELETE /api/minMaxDataElements.json
```

The JSON content is in similar format as above:

```
{
    "min": 1,
    "generated": false,
    "max": 100,
    "dataElement": {
        "id": "UOlfIjgN8X6"
    },
    "source": {
        "id": "DiszpKrYNg8"
    },
    "optionCombo": {
        "id": "psbwp3CQEhs"
    }
}
```

Lock exceptions

The lock exceptions resource allows you to open otherwise locked data sets for data entry for a specific data set, period and organisation unit. You can read lock exceptions from the following resource:

```
/api/lockExceptions
```

To create a new lock exception you can use a POST request and specify the data set, period and organisation unit:

```
POST /api/lockExceptions?ds=BfMAe6Itzgt&pe=201709&ou=DiszpKrYNg8
```

To delete a lock exception you can use a similar request syntax with a DELETE request:

```
DELETE /api/lockExceptions?ds=BfMAe6Itzgt&pe=201709&ou=DiszpKrYNg8
```

Tokens

The *tokens* resource provides access tokens to various services.

Google Service Account

You can retrieve a Google service account OAuth 2.0 access token with a GET request to the following resource.

```
GET /api/tokens/google
```

The token will be valid for a certain amount of time, after which another token must be requested from this resource. The response contains a cache control header which matches the token expiration. The response will contain the following properties in JSON format.

Token response

Property	Description
access_token	The OAuth 2.0 access token to be used when authentication against Google services.
expires_in	The number of seconds until the access token expires, typically 3600 seconds (1 hour).
client_id	The Google service account client id.

This assumes that a Google service account has been set up and configured for DHIS2. Please consult the installation guide for more info.

Analytics table hooks

Analytics table hooks provide a mechanism for invoking SQL scripts during different phases of the analytics table generation process. This is useful for customizing data in resource and analytics tables, e.g. in order to achieve specific logic for calculations and aggregation. Analytics table hooks can be manipulated at the following API endpoint:

```
/api/analyticsTableHooks
```

The analytics table hooks API supports the standard HTTP CRUD operations for creating (POST), updating (PUT), retrieving (GET) and deleting (DELETE) entities.

Hook fields

Analytics table hooks have the following fields:

Analytics table hook fields

Field	Options	Description
name	Text	Name of the hook.
phase	RESOURCE_TABLE_POPULATED, ANALYTICS_TABLE_POPULATED	The phase for when the SQL script should be invoked.
resourceTableType	See column "Table type" in table "Phases, table types and temporary tables" below	The type of resource table for which to invoke the SQL script. Applies only for hooks defined with the RESOURCE_TABLE_POPULATED phase.
analyticsTableType	See column "Table type" in table "Phases, table types and temporary tables" below	The type of analytics table for which to invoke the SQL script. Applies only for hooks defined with the ANALYTICS_TABLE_POPULATED phase.
sql	Text	The SQL script to invoke.

The *ANALYTICS_TABLE_POPULATED* phase takes place after the analytics table has been populated, but before indexes have been created and the temp table has been swapped with the main table. As a result, the SQL script should refer to the analytics temp table, e.g. *analytics_temp*, *analytics_completeness_temp*.

This applies also to the *RESOURCE_TABLE_POPULATED* phase, which takes place after the resource table has been populated, but before indexes have been created and the temp table has been swapped with the main table. As a result, the SQL script should refer to the resource temp table, e.g. *_orgunitstructure_temp*, *_categorystructure_temp*.

You should define only one of the *resourceTableType* and *analyticsTableType* fields, depending on which *phase* is defined.

You can refer to the temporary database table which matches the specified hook table type only (other temporary tables will not be available). As an example, if you specify *ORG_UNIT_STRUCTURE* as the resource table type, you can refer to the *_orgunitstructure_temp* temporary database table only.

The following table shows the valid combinations of phases, table types and temporary tables.

Phases, table types and temporary tables

Phase	Table type	Temporary table
RESOURCE_TABLE_POPULATED	ORG_UNIT_STRUCTURE	_orgunitstructure_temp
	DATA_SET_ORG_UNIT_CATEGORY	_datasetorgunitcategory_temp
	CATEGORY_OPTION_COMBO_NAME	_categoryoptioncomboname_temp
	DATA_ELEMENT_GROUP_SET_STRUCTURE	_dataelementgroupsetstructure_temp
	INDICATOR_GROUP_SET_STRUCTURE	_indicatorgroupsetstructure_temp
	ORG_UNIT_GROUP_SET_STRUCTURE	_organisationunitgroupsetstructure_temp

Phase	Table type	Temporary table
	CATEGORY_STRUCTURE	_categorystructure_temp
	DATA_ELEMENT_STRUCTURE	_dataelementstructure_temp
	PERIOD_STRUCTURE	_periodstructure_temp
	DATE_PERIOD_STRUCTURE	_dateperiodstructure_temp
	DATA_ELEMENT_CATEGORY_OPTION_COMBO	_dataelementcategoryoptioncombo_temp
	DATA_APPROVAL_MIN_LEVEL	_dataapprovalminlevel_temp
ANALYTICS_TABLE_POPULATED	DATA_VALUE	analytics_temp
	COMPLETENESS	analytics_completeness_temp
	COMPLETENESS_TARGET	analytics_completenesstarget_temp
	ORG_UNIT_TARGET	analytics_orgunittarget_temp
	EVENT	analytics_event_temp_<program-uid>
	ENROLLMENT	analytics_enrollment_temp_<program-uid>
	VALIDATION_RESULT	analytics_validationresult_temp

Creating hooks

To create a hook which should run after the resource tables have been populated you can do a *POST* request like this using *JSON* format:

```
curl -d @hooks.json "localhost/api/analyticsTableHooks" -H "Content-Type:application/json" -u
admin:district
```

```
{
    "name": "Update 'Area' in org unit group set resource table",
    "phase": "RESOURCE_TABLE_POPULATED",
    "resourceTableType": "ORG_UNIT_GROUP_SET_STRUCTURE",
    "sql": "update _organisationunitgroupsetstructure_temp set \"uIuxlbV1vRT\" = 'b0EsAxm8Nge'"
}
```

To create a hook which should run after the data value analytics table has been populated you can do a *POST* request like this using *JSON* format:

```
{
    "name": "Update 'Currently on treatment' data in analytics table",
    "phase": "ANALYTICS_TABLE_POPULATED",
    "analyticsTableType": "DATA_VALUE",
    "sql": "update analytics_temp set monthly = '200212' where \"monthly\" in ('200210', '200211')"
}
```

Metadata repository

DHIS2 provides a metadata repository containing metadata packages with various content. A metadata package is a DHIS2-compliant JSON document which describes a set of metadata objects.

To retrieve an index over available metadata packages you can issue a GET request to the *metadataRepo* resource:

```
GET /api/synchronization/metadataRepo
```

A metadata package entry contains information about the package and a URL to the relevant package. An index could look like this:

```
{
    "packages": [
        {
            "id": "sierre-leone-demo",
            "name": "Sierra Leone demo",
            "description": "Sierra Leone demo database",
            "version": "0.1",
            "href": "https://dhis2.org/metadata-repo/221/sierra-leone-demo/metadata.json"
        },
        {
            "id": "trainingland-org-units",
            "name": "Trainingland organisation units",
            "description": "Trainingland organisation units with four levels",
            "version": "0.1",
            "href": "https://dhis2.org/metadata-repo/221/trainingland-org-units/metadata.json"
        }
    ]
}
```

A client can follow the URLs and install a metadata package through a POST request with content type *text/plain* with the metadata package URL as the payload to the *metadataPull* resource:

```
POST /api/synchronization/metadataPull
```

An example curl command looks like this:

```
curl "localhost:8080/api/synchronization/metadataPull" -X POST
  -d "https://dhis2.org/metadata-repo/221/trainingland-org-units/metadata.json"
  -H "Content-Type:text/plain" -u admin:district
```

Icons

DHIS2 includes a collection of icons that can be used to give visual context to metadata. These icons can be accessed through the icons resource.

```
GET /api/icons
```

This endpoint returns a list of information about the available icons. Each entry contains information about the icon, and a reference to the actual icon.

```
{
    "key": "mosquito_outline",
    "description": "Mosquito outline",
    "keywords": ["malaria", "mosquito", "dengue"],
```

```
    "href": "<dhis server>/api/icons/mosquito_outline/icon.svg"
}
```

The keywords can be used to filter which icons to return. Passing a list of keywords with the request will only return icons that match all the keywords:

```
GET /api/icons?keywords=shape,small
```

A list of all unique keywords can be found at the keywords resource:

```
GET /api/icons/keywords
```