DHIS2 Developer Manual

2.25

Table of Contents

Chapter 1. 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.

1.1. 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 a Accept: application/pdf header or append .pdf or ?format=pdf to your request URL.

3. Interactions with the API requires 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.

1.2. 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/25/me
```

1.2.1. 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 frameworks provides support for authentication using Basic, such as Apache HttpClient, Spring RestTemplate and C# WebClient. An important note is that this authentication scheme provides no security since the username and password is sent in plain text and can be easily decoded. Using it is recommended only if the server is using SSL/TLS (HTTPS) to encrypt communication between itself and the client. Consider it a hard requirement to provide secure interactions with the Web API.

1.2.2. OAuth2

DHIS2 supports the OAuth2 authentication protocol. OAuth2 is an open standard for authorization which it 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. DHIS 2 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, click add new and enter the desired client name and the grant types.

1.2.2.1. 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:

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

```
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.

1.2.2.2. Grant type password

The simplest of all grant types is the password grant type. This grant type is similar to basic authenticaion in the sense that it requires the client to collect the users 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/25/
dataElements.json
```

1.2.2.3. 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 roundtrip 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 an token to start with.

1.2.2.4. 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 redirect_uris part of the client payload.

Step 1: Using a browser visit this URL (if you have more than one redirect URIs, you might want to add &redirect_uri=http://www.example.org) :

```
SERVER="https://play.dhis2.org/dev"

$SERVER/uaa/oauth/authorize?client_xml:id=demo&response_type=code
```

Step 2: After the user have 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 a 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/token
  -d grant_type=authorization_code -d code=XYZ
```

1.3. 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.

Table 1.1. WebMessage properties

Name	Description
httpStatus	HTTP Status message for this response, see RFC 2616 (Section 10) for more information.
httpStatusCode	HTTP Status code for this response, see RFC 2616 (Section 10) for more information.
status	DHIS2 status, possible values are OK	WARNING	ERROR, where OK is means everything was successful, ERROR means that operation did not complete and WARNING means operation was partially successful, if there 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 techincal 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.

1.4. 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.

Table 1.2. Period format

Interval	Format	Example	Description
Day	yyyyMMdd	20040315	March 15 2004
Week	yyyyWn	2004W10	Week 10 2004
Month	yyyyMM	200403	March 2004
Quarter	yyyyQn	2004Q1	January-March 2004
Six-month	yyyySn	2004S1	January-June 2004
Six-month April	yyyyAprilSn	2004AprilS1	April-September 2004
Year	yyyy	2004	2004
Financial Year April	yyyyApril	2004April	Apr 2004-Mar 2005

Interval	Format	Example	Description
Financial Year July	yyyyJuly	2004July	July 2004-June 2005
Financial Year Oct	yyyyOct	2004Oct	Oct 2004-Sep 2005

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 allows 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
```

1.5. Identifier Schemes

Throughout the Web API we refer to different kinds of id schemes, this section will give a explanation to how they can be used. Please note that not all schemes works for all web-api calls, and not not all schemes can be used for both input and output (this is outlined in the sections explaining the web-apis themself). Setting the idScheme allows you to override the default for all schemes, but if other schemes are declared like dataElementIdScheme this this will be used instead.

The full range of scheme parameters available are:

```
idScheme, dataElementIdScheme, categoryOptionComboIdScheme, orgUnitIdScheme,
programIdScheme, programStageIdScheme, trackedEntityIdScheme,
 trackedEntityAttributeIdScheme
```

The default scheme for all parameters are what we call the stable identifiers i.e. UIDs, but we also have different schemes depending on how you want object matching to behave.

Table 1.3. 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 not that this uses what is available as object.name, and not the translated name. Also not 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.

1.6. Browsing the Web API

The entry point for browsing the Web API is /api/. This resource provide 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:

Table 1.4. 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/25/dataElementGroups.xml?links=false&paging=false
```

You can query for elements on the name property instead of returning 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/25/dataElements?query=anaemia
```

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

```
/api/25/dataElements.json?page=2&pageSize=20
```

You can completely disable paging like this:

```
/api/25/indicatorGroups.json?paging=false
```

To order the result based on a specific property:

```
/api/25/indicators.json?order=shortName:desc
```

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

```
/api/25/identifiableObjects/<id>
```

1.6.1. 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 includes displayName, displayShortName and displayDescription.

Table 1.5. Translate options

Parameter	Values	Description
translate	true | false	Translate display* properties in metadata output (displayName, displayShortName, displayDescription,

Parameter	Values	Description
		and displayFormName for data elements). Default value is true.
locale	Locale to use	Translate metadata output using a specified locale (requires translate=true).

1.6.2. 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 are 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"
      },
      {
        "property": "SHORT_NAME",
        "locale": "en_FK",
        "value": "aa"
      },
      {
        "property": "NAME",
        "locale": "en_GB",
        "value": "Accute Flaccid Paralysis (Deaths < 5 yrs)"
      }
    ]
}
```

Note that even if you get the unfiltered result, and are using the appropriate type endpoint i..e /api/25/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/25/<object-type>/<object-id>/translations endpoint. As an example, for a data element with identifier FTRrcoaog83 you could use /api/25/dataElements/FTRrcoaog83/translations to get and update translations. The fields available are property with options NAME, SHORT_NAME, DESCRIPTION, the locale which supports any valid locale ID and the the value itself.

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 an data element with ID FTRrcoaog83 you can PUT this to /api/25/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).

1.7. Working with the metadata API

The metadata resource can be accessed at /api/metadata. This resource lets you read and write the full set of metadata. This section will give a basic introduction to working with this API. For specific synchronization issues, please see the integration chapter.

By default, interacting with /api/metadata using the GET HTTP method will give you all metadata rendered as XML. You can also be more specific about the metadata elements you are interested in.

1.7.1. Content types

The Web API offers several content types for metadata.

Table 1.6. Available Content-Types

Content-Type	URL extension	Description
application/xml	.xml	Returns the metadata in XML representation
application/json	.json	Returns the metadata in JSON representation
application/pdf	.pdf	Returns the metadata as a PDF document
application/csv	.csv	Returns the metadata as a CSV file
application/ vnd.ms-excel	.xls | .xlsx	Returns the metadata as an Excel workbook

1.7.2. Query parameters

The following query parameters are available for customizing your request.

Table 1.7. Available Query Filters

Param	Type	Required	Options (default first)	Description
assumeTrue	boolean	false	true | false	Indicates whether to get all resources or no resources by default.
viewClass	enum	false	export | basic | detailed	Alternative views of the metadata. Please note that only metadata exported with viewClass=export or detailed can be used for import.
dryRun	boolean	false	false | true	If you set this to true, the actual import will not happen. Instead the system will generate a summary of what would have been done.
{resources}	boolean	false	true | false (default	See /api for available resources.

Param	Type	Required	Options (default first)	Description
			depends on assumeTrue)	Indicates which resources to include in the response.
lastUpdated	date	false	Several formats are available: yyyy, yyyy-MM, yyyy-MM-dd, yyyyMM, yyyyMMdd	Filters the metadata based on the lastUpdated field.
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.
sharing	boolean	false	false | true	Should sharing be supported or not. The default is false, which is the old behavior. You can set this to true to allow updating user, publicAccess and userGroupAccesses fields (if not they are cleared out on create,

Param	Type	Required	Options (default first)	Description
				and not touched on update).
mergeMode	enum	false	REPLACE, MERGE	Strategy for merging of objects when doing updates. REPLACE will just overwrite the propery with the new value provided, MERGE will only set the property if its not null (only if the property was provided).
async	boolean	false	false | true	Indicates wether the import should be done async or not, the default is false which means the client will wait until the import is done, this is probably what you want for small imports (as you will get the import summary directly back to you). For large imports, the client might time out, so async=true is recommended, and the client connection will be dropped when the payload is uploaded.

1.7.3. Available strategies for import

When importing data using the metadata resource you can define various strategies for import.

Table 1.8. Available Strategies

Type	Description
CREATE_AND_UPDATE	Allows creation and updating of objects.
CREATE	Allows creation of objects only.
UPDATE	Allows update of objects only.
DELETE	Allows deletes of objects only.

1.7.4. Metadata import examples

Example: Get a filtered set of metadata that was updated since August 1, 2014

As described in the last section, there is a number of options you can apply to /api/metadata to give you a filtered view. The use-case we will be looking into here is where you want a nightly job that synchronizes organisation units. We will be using cURL as the HTTP client.

```
curl -H "Accept: application/xml" -u admin:district
  "https://play.dhis2.org/demo/api/metadata?
assumeTrue=false&organisationUnits=true&lastUpdated=2014-08-01"
```

Example: Get metadata that was updated since February 2014

This example will just the default assumeTrue setting, along with getting the last updates from February 2014. This means that every single type that has been updated will be retrieved.

```
curl -H "Accept: application/xml" -u admin:district "https://play.dhis2.org/demo/api/
metadata?lastUpdated=2014-02"
```

Example: Create metadata

The metadata resource can also be used to create or update metadata by using the POST HTTP method. The metadata content can be both XML and JSON, using "application/xml" and "application/json" content type respectively. The request payload content will be accepted in several formats, including plain text, zipped and gzipped. POSTing a metadata payload can be done for example like this, where metadata.xml is a file in the same directory with the metadata content:

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

The import will happen in a asyncronous process which implies that the response will return as soon as the process is started. The response status code to be expected is 204 No Content.

1.8. 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, are supported as well.

Table 1.9. Available Operators

Operator	Types	Value required	Description
eq	string | boolean | integer | float | enum	true	Equality

Operator	Types	Value required	Description
	| collection (checks for size) | date		
!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
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 |	true	Less than

Operator	Types	Value required	Description
	collection (checks for size) | date		
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
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. The filtering mechanism allows for recursion. See below for some examples.

Get data elements with id property ID1 or ID2:

```
/api/25/dataElements?filter=id:eq:ID1&filter=id:eq:ID2
```

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

```
/api/25/dataElements?filter=dataSets.id:eq:ID1
```

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

```
/api/25/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/25/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/25/dataElements.json?filter=id:in:[fbfJHSPpUQD,cYeuwXTCPkU]
```

1.9. Metadata field filter

In certain 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 is very simple and 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 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 (see below). Both XML and JSON are supported.

Example: Get id and name on the indicators resource:

```
/api/25/indicators?fields=id,name
```

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

```
/api/25/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.

Table 1.10. 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, also works inside objects/collections. Useful when you use a preset to inlude 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 table below for descriptions.

Table 1.11. 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 a object, does not take into consideration if the object is the owner of the relation.
owner	Returns all persisted property on a object where the object is the owner of all properties, this payload can be used to update through the web-api.

Example: Include all fields from dataSets except organisationUnits:

```
/api/25/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/25/dataSets/BfMAe6Itzgt?fields=id,name,organisationUnits[:all,!id]
```

Example: Include nameable properties from all indicators:

```
/api/25/indicators.json?fields=:nameable
```

1.9.1. Field transformers

In DHIS2.17 we introduced field transformers, the idea is to allow further customization of the properties on the server side. For 2.17 we only supports one transformer called rename, it can be used like this:

```
/api/25/dataElements/ID?fields=id|rename(i),name|rename(n)
```

This will rename the id property to i and name property to n. Please note that the format should be considered beta in 2.17, and the format might be changed in 2.18.

1.9.2. Field converters

In DHIS2.17 alongside transformers we also introduced field converters, while field transformers usually do minor changes to the data stream (name changes etc), field converters can completely change the output of the data. For 2.17 we are including 3 field converters:

Table 1.12. Field converters

Name	Description
size	Gives sizes of strings (length) and collections, i.e. /api/dataElements?fields=dataSets::size
isEmpty	Is string or collection empty, i.e. /api/dataElements?fields=dataSets::isEmpty
isNotEmpty	Is string or collection not empty, i.e. /api/dataElements?fields=dataSets::isNotEmpty

1.10. Metadata create, read, update, delete, validate

While some of the web-api endpoints already contains support for CRUD (create, read, update, delete), from version 2.15 this is now supported on all endpoints. It should work as you expect, and the subsections will give more detailed information about create, update, and delete (read is already covered elsewhere, and have been supported for a long time).

1.10.1. Create / update parameters

The following query parameters are available for customizing your request.

Table 1.13. 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

Param	Type	Required	Options (default first)	Description
				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 propery with the new value provided, MERGE will only set the property if its not null (only if the property was provided).

1.10.2. 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 an 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 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/25/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/25/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/25/constants/ID
```

1.10.3. Deleting objects

Deleting objects are very straight forward, you will need to know the ID and the endpoint of the type you want 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/25/constants/ID
```

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

1.10.4. Adding and removing objects to/from collections

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

```
/api/25/{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. 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/25/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/25/categories/IDA/categoryOptions/IDB
```

If you want to add or remove multiple items from a collection in one request that is also available using a payload like this:

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

Using this payload, you have 3 options, you can delete, add, or replace.

Adding Items:

```
POST /api/25/categories/IDA/categoryOptions
```

Replacing Items:

```
PUT /api/25/categories/IDA/categoryOptions
```

Delete Items:

```
DELETE /api/25/categories/IDA/categoryOptions
```

1.10.5. Validating payloads

System wide validation of metadata payloads are enabled from 2.19 release, this means that create/update operations on the web-api endpoints will be checked for valid payload before allowed changes to be made, to find out what validations are in place for a endpoint, please 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
{ payload }
```

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."
    }
```

```
]
```

1.10.6. 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, both for single properties and for multiple properties.

The format for updating a single property is the same as when you are updating a complete object, just with only 1 property in the JSON/XML file, i.e.:

```
curl -X PATCH -d "{\"name\": \"New Name\"}" -H "Content-Type: application/json"
-u admin:district https://play.dhis2.org/dev/api/25/dataElements/fbfJHSPpUQD/name
```

Please note that we are including the property name two times, one time in the payload, and one time in the endpoint, the generic endpoint for this is /api/type/id/property-name, and the Content-Type must also be included as usual (since we support multiple formats).

The format for updating multiple properties are similar, just that we don't include the property names in the url, i.e.:

```
{ // file.json
  "name": "Updated Name",
  "zeroIsSignificant": true
}
```

```
curl -X PATCH -d @file.json -H "Content-Type: application/json"
-u admin:district https://play.dhis2.org/dev/api/25/dataElements/fbfJHSPpUQD
```

1.11. Metadata export

This section explains the metatada API which is available at /api/23/metadata and /api/25/metadataendpoints. XML and JSON resource representations are supported.

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 export of certain types by setting type=true/false.

Table 1.14. 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.

1.11.1. Metadata export examples

Export all metadata:

```
curl -u user:pass http://server/api/25/metadata
```

Export all metadata ordered by lastUpdated descending:

```
curl -u user:pass http://server/api/25/metadata?defaultOrder=lastUpdated:desc
```

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

```
curl -u user:pass http://server/api/25/metadata?
dataElements:fields=id,name&dataElements:order=displayName:desc
```

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

```
curl -u user:pass http://server/api/25/metadata?
defaultFilter=name:^like:ANC&dataElements=true&indicators=true
```

1.11.2. Data set and program export with dependencies

When you want to move a whole set of data set or program metadata from one server to
another (possibly empty) server, we have two special endpoints for just that purpose:

```
/api/25/dataSets/ID/metadata.json
```

```
/api/25/programs/ID/metadata.json
```

These exports can be imported both using the old importer at /api/metadata and the new
importer at /api/23/metadata.

1.12. Metadata import

This section explains the metatada API which is available at /api/23/metadata and /api/25/
metadataendpoints. XML and JSON resource representations are supported.

The importer allows you to import metadata exported with the new exporter. The various
parameters are listed below.

Table 1.15. 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 have 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.
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	MERGE, REPLACE	Sets the merge mode, when doing updates we have two ways of merging the old object with the new one,

Name	Options (first is default)	Description
		MERGE mode will only overwrite the old property if the new one is not-null, for REPLACE mode all properties are overwritten regardsless of null or not.
flushMode	AUTO, OBJECT	Sets the flush mode, decides when to flush the internal cache, strongly reommended 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).
skipSharing	false, true	Skip sharing properties, does not merge sharing when doing updates, and does not add user group accessses when creating new objects.
skipValidation	false, true	Skip validation for import. NOT RECOMMENDED.

1.13. CSV metadata import

DHIS2 supports import of metadata in the CSV format. Columns which are not required can be omitted in the CSV file, but the order will be affected. If you would like to specify columns which appear late in the order but not specify columns which appear early in the order you can include empty columns ("") for them. The following object types are supported:

- Data elements
- Data element groups
- Category options
- Category option groups
- Organisation units
- Organisation unit groups
- Validation rules
- Translations
- Option sets

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

1.13.1. Data elements

Table 1.16. Data Element CSV Format

Column	Required	Value (default first)	Description
Name	Yes		Name. Max 230 char. Unique.
UID	No	UID	Stable identifier. Exactly 11 alpha-numeric characters, beginning with a character. Will be generated by system if not specified.
Code	No		Stable code. Max 50 char.
Short name	No	50 first char of name	Will fall back to first 50 characters of name if unspecified. Max 50 char. Unique.
Description	No		Free text description.

Column	Required	Value (default first)	Description
Form name	No		Max 230 char.
Domain type	No	AGGREGATE | TRACKER	Domain type for data element, can be aggregate or tracker. Max 16 char.
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.
Aggregation operator	No	SUM | AVERAGE | AVERAGE_SUM_ORG_UNIT | COUNT | STDDEV | VARIANCE | MIN | MAX | NONE	Operator indicating how to aggregate data in the time dimension. Max 16 char.
Category combination UID	No	UID	UID of category combination. Will default to default category combination if not specified.
Url	No		URL to data element resource. Max 255 char.
Zero is significant	No	false | true	Indicates whether zero values will be stored for this data element.
Option set	No	UID	UID of option set to use for data.
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 in skill development training",,"D0001","Women participated
 development training"
"Women participated in community organizations",,"D0002","Women participated community
 organizations"
```

1.13.2. Organisation units

Table 1.17. Organisation Unit CSV Format

Column	Required	Value (default first)	Description
Name	Yes		Name. Max 230 characters. Unique.
UID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.

Column	Required	Value (default first)	Description
Code	No		Stable code. Max 50 char.
Parent UID	No	UID	UID of parent organisation unit.
Short name	No	50 first char of name	Will fall back to first 50 characters of name if unspecified. Max 50 characters. Unique.
Description	No		Free text description.
UUID	No		UUID. Max 36 char.
Opening date	No	1970-01-01	Opening date of organisation unit in YYYY-MM-DD format.
Closed date	No		Closed date of organisation unit in YYYY-MM-DD format, skip if currently open.
Comment	No		Free text comment for organisation unit.
Feature type	No	NONE | MULTI_POLYGON | POLYGON | POINT | SYMBOL	Geospatial feature type.
Coordinates	No		Coordinates used for geospatial analysis in Geo JSON format.
URL	No		URL to organisation unit resource. Max 255 char.
Contact person	No		Contact person for organisation unit. Max 255 char.
Address	No		Address for organisation unit. Max 255 char.
Email	No		Email for organisation unit. Max 150 char.
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"
```

1.13.3. Validation rules

Table 1.18. Validation Rule CSV Format

Column	Required	Value (default first)	Description
Name	Yes		Name. Max 230 characters. Unique.
UID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.
Code	No		Stable code. Max 50
Description	No		Free text description.
Instruction	No		Free text instruction.
Importance	No	MEDIUM | HIGH | LOW	
Rule type	No	VALIDATION | SURVEILLANCE	

Column	Required	Value (default first)	Description
Operator	No	equal_to | not_equal_to | greater_than | greater_than_or_equal_to | less_than | less_than_or_equal_to | compulsory_pair	
Period type	No	Monthly | Daily | Weekly | Quarterly | SixMontly | Yearly	
Left side expression	Yes		Mathematical formula based on data element and option combo UIDs.
Left side expression description	Yes		Free text.
Left side null if blank	No	false | true	Boolean.
Right side expression	Yes		Mathematical formula based on data element and option combo UIDs.
Right side expression description	Yes		Free text.
Right side null if blank	No	false | true	Boolean.

1.13.4. Translations

Table 1.19. Translation CSV format

Column	Required	Value	
ObjectId	Yes	UID	UID of object to translate.
ClassName	Yes		Name of class to translate, e.g. DataElement, IndicatorGroupSet.
Locale	Yes		Locale value, e.g. "es", "fr".
Property	Yes	name | shortName | description	Property to translate.
Value	Yes		Translation.

An example of the translations CSV format looks like this:

```
objectid,classname,locale,property,value
"s46m5MS0hxu","DataElement","fr","name","Doses bcg donnés"
"UOlfIjgN8X6","DataElement","fr","name","Enfant complètement vacciné"
```

1.13.5. Option sets

Table 1.20. Option Set CSV Format

Column	Required	Value (default first)	Description
OptionSetName	Yes		Name. Max 230 characters. Unique. Should be repeated for each option.
OptionSetUID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified. Should be repeated for each option.
OptionSetCode	No		Stable code. Max 50 char. Should be repeated for each option.
OptionName	Yes		Option name. Max 230 characters.
OptionUID	No	UID	Stable identifier. Max 11 char. Will be generated by system if not specified.
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",,,"Uknown",,"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"
```

1.13.6. Other objects

Table 1.21. Data Element Group, Category Option, Category Option Group, Organisation Unit Group CSV Format

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

An example for category options looks like this:

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

```
"Female",,"FEMALE"
```

1.14. 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/25/fileResources
```

The contents of a file resources is not directly accessible but is 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 -X POST -v -F "file=@/Path/to/file;filename=name-of-file.png" https://server/
api/25/fileResources
```

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 (this is handled for you by any Web browser) but will be replaced by 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 Section 1.17.7.1, "Working with file data values"). If we need to check the updated storageStatus or otherwise retrieve the meta-data of the file, the fileResources endpoint can be queried.

```
curl -v https://server/api/25/fileResources/xm4JwRwke0i -H "Accept: application/json"
```

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

1.14.1. 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.

1.15. Metadata versioning

This section explains the Metadata Versioning APIs available starting 2.24

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

Table 1.22. 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>"

1.15.1. Get metadata version details

Example: Get the current metadata version of this system

Sample request:

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

Sample 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"

Sample request:

```
curl -u admin:district "https://play.dhis2.org/dev/api/metadata/version?
versionName=Version_2"
```

Sample 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 api will return the list of all metadata versions of the
  system on which it is invoked.

Table 1.23. 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.

1.15.2. Get list of all metadata versions

Example: Get the list of all versions in this system

Sample request:

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

Sample 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"

Sample request:

```
curl -u admin:district "https://play.dhis2.org/dev/api/metadata/version/history?
baseline=Version_2"
```

Sample 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 api will create the metadata version for the version type as specified in the parameter.

Table 1.24. 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 - 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 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 better understanding.

1.15.3. Create metadata version

Example: To create metadata version of type BEST_EFFORT

Sample request:

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

Sample 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 api will download the actual metadata specific to the version name passed as path parameter.

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

Table 1.25. Path parameters

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

1.15.4. Download metadata

Example: Get the actual metadata for "Version_5"

Sample request:

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

Sample 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"
        }
    }]
}
```

1.16. Metadata Synchronization

This section explains the Metadata Synchronization API available starting 2.24

- /api/metadata/sync - This api performs a 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.

Table 1.26. 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 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 api 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 this API only.

1.16.1. Sync metadata version

Example: Sync Version_6 from central system to this system

Sample request:

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

1.17. Data values

This section is about sending and reading data values.

```
/api/25/dataValueSets
```

1.17.1. 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 and we recommend that you follow the provided links with a web browser while reading (log in with admin/district as username/password). 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 national DHIS2 instance.

The resource which is most appropriate for our purpose of sending data values is the dataValueSets resource. A data value set represents a set of data values which have a logical relationship, usually from being captured off the same data entry form. We follow the link to the HTML representation which will take us to http://play.dhis2.org/demo/api/24/dataValueSets. The format looks like this:

```xml
<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:

```json
{
  "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","storedby","lastupd","comme
"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 return to the entry point at http://play.dhis2.org/demo/api/24 and follow the embedded link pointing at the dataSets resource located at http://

play.dhis2.org/demo/api/24/dataSets. From there we find and follow the link to the Mortality < 5 years data set which leads us to http://play.dhis2.org/demo/api/24/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 facility (org unit). The dataSet representation conveniently provides link to org units which report on it so we search for Ngelehun CHC and follow the link to the HTML representation at http://play.dhis2.org/demo/api/24/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",
  "completeData": "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/25/dataValueSets"
  -H "Content-Type:application/xml" -u admin:district -v
```

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

```
curl -d @datavalueset.json "https://play.dhis2.org/demo/api/25/dataValueSets"
  -H "Content-Type:application/json" -u admin:district -v
```

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.

1.17.2. 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 with http://play.dhis2.org/demo/api/24/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 org 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/25/dataValueSets"
  -H "Content-Type:application/xml" -u admin:district -v
```

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 -v
```

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.

1.17.3. Import parameters

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

Table 1.27. 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.
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.
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

Parameter	Values (default first)	Description
		performance overhead. The latter is faster but requires the connection to persist until the process is finished.

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

```
/api/25/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 huge impact for performance. For small import files, leaving it to false will be fast. For large import files which contain a large number of distint data elements and organisation units, setting it to true will be orders of magnitude faster.

1.17.3.1. Identifier schemes

Regarding the id schemes, by default the identifiers used in the XML messages uses the DHIS2 stable object identifiers referred to as uid. In certain interoperability situations we might experience that 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 identfier 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 availabe.

- 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 uid of the attribute, e.g. "attributeDnrLSdo4hMl".

1.17.4. 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.

Table 1.28. 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","categoryoptioncombo","attroptioncombo","value","storedby","time
"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"
```

1.17.5. 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/25/dataSets/BfMAe6Itzgt/dataValueSet.json
```

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

Table 1.29. 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.

1.17.6. 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:

Table 1.30. 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.
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.
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 object to use for data values in response.

The following response formats are supported:

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

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/25/dataValueSets?
dataSet=pBOMPrpg1QX&period=201401&orgUnit=DiszpKrYNg8"
  -H "Accept:application/xml" -u admin:district -v
```

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 presedence over the start and end date parameters. An exampe looks like this:

```
curl "https://play.dhis2.org/demo/api/25/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 -v
```

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

```
https://play.dhis2.org/demo/api/25/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:

```
https://play.dhis2.org/demo/api/25/dataValueSets.json?
dataSet=pBOMPrpg1QX&period=201401&orgUnit=DiszpKrYNg8
```

The response will look something like this:

```
{
  "dataSet": "pBOMPrpg1QX",
  "completeData": "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" }
  ]
}
```

You can also request data in CSV format like this:

```
https://play.dhis2.org/demo/api/25/dataValueSets.csv?
dataSet=pBOMPrpg1QX&period=201401&orgUnit=DiszpKrYNg8
```

The response will look like this:

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

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.

1.17.7. 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:

```
https://play.dhis2.org/demo/api/25/dataValues
```

The following query parameters are supported for this resource:

Table 1.31. 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 combine with cp)	Attribute combo identifier
cp	No (must combine with cc)	Attribute option identifiers, separated with ; for multiple values
value	No	Data value
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 are invalid or if the data is locked, the response will contain the 409 Conflict status code and descriptive text message. If the operation lead 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/25/dataValues?de=s46m5MS0hxu
  &pe=201301&ou=DiszpKrYNg8&co=Prlt0C1RF0s&value=12"
  -X POST -u admin:district -v
```

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 combination, together with the identifier(s) of the attribute option(s) which the value represents within the combination. An example looks like this:

```
curl "https://play.dhis2.org/demo/api/25/dataValues?de=s46m5MS0hxu&ou=DiszpKrYNg8
  &pe=201308&cc=dzjKKQq0cSO&cp=wbrDrL2aYEc;btOyqprQ9e8&value=26"
  -X POST -u admin:district -v
```

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/25/dataValues?de=s46m5MS0hxu
  &pe=201301&ou=DiszpKrYNg8&co=Prlt0C1RF0s"
  -X GET -u admin:district -v
```

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

1.17.7.1. 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/25/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 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 dataValues/files endpoint. This is especially true for large files which might require time consuming uploads happening in the background to a 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.

1.18. 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 (Integerating 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>
```

1.18.1. The adx root element

The adx root element has only one manadatory 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.

1.18.2. 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 identifiier 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 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

1.18.3. Data values

The dataValue element in adx is very similar to its equivalent in DXF. The mandatory attributes are dataElement and value. 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 disaggregation of data. In adx the disaggregations (eg 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 chracters 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 synched 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.

1.18.4. Importing data - HTTP POST

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/xml+adx"
-d @data.xml "https://play.dhis2.org/demo/api/25/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.

1.18.5. Exporting data - HTTP GET

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 -X POST -H "Accept: application/xml+adx"
 "https://play.dhis2.org/demo/api/25/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.

1.19. Events

This section is about sending and reading events.

```
/api/25/events
```

1.19.1. 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
<?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/25/events"
  -H "Content-Type:application/xml" -u admin:district -v
```

The same payload in JSON format looks like this:

```json
{
  "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" },
    { "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/25/events" -H "Content-Type:application/json" -u
 admin:district -v
```

We also support sending multiple events at the same time. A payload in XML format might look like this:

```xml
<?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:

```json
{
  "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" }
    ] }
```

```
  ]
}
```

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.

Table 1.32. Events resource format

Parameter	Type	Required	Options (default first)
program	string	true	
orgUnit	string	true	
eventDate	date	true	
status	enum	false	ACTIVE | COMPLETED | VISITED | SCH | SKIPPED
storedBy	string	false	Defaults to current user
coordinate	double	false	
dataElement	string	true	
value	string	true	

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.

Update: 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.

```
curl -X PUT -d @updated_event.xml "localhost/api/25/events/ID"
  -H "Content-Type: application/xml" -u admin:district
```

```
curl -X PUT -d @updated_event.json "localhost/api/25/events/ID"
  -H "Content-Type: application/json" -u admin:district
```

Delete: To delete an existing event, all you need is to send a DELETE request with a identifier reference to the server you are using.

```
curl -X DELETE "localhost/api/25/events/ID" -u admin:district
```

Get: To get an existing event you can issue a GET request including the identifier like this:

```
curl "localhost/api/25/events/ID" -H "Content-Type: application/xml" -u admin:district
```

1.19.1.1. Import parameters

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

Table 1.33. 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

Parameter	Values (default first)	Description
		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.
importStrategy	CREATE | UPDATE | CREATE_AND_UPDATE | DELETE	Save objects of all, new or update import status on the server.
sendNotifications	true | false	Indicates whether to send notifications for completed events.

1.19.2. 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:

Table 1.34. 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 stage 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
10	longitude	double	Longitude where event happened
11	dataElement	identifier	Identifier of data element

Index	Key	Type	Description
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

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,,
```

1.19.3. 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.

Table 1.35. 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

Key	Type	Required	Description
			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
lastUpdated	date	false	Filter for events which were updated after this date.
skipMeta	boolean	false	Exclude the meta data part of response (improves performance)
page	integer	false	Page number
pageSize	integer	falase	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 and CODE
categoryOptionComboIdScheme	string	false	Category Option Combo ID scheme to use for export, valid options are UID and CODE
orgUnitIdScheme	string	false	Organisation Unit ID scheme to use for export, valid options are UID and CODE
programIdScheme	string	false	Program ID scheme to use for export, valid options are UID and CODE
programStageIdScheme	string	false	Program Stage ID scheme to use for

Key	Type	Required	Description
			export, valid options are UID and CODE
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 retreive 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
event	comma delimited strings	false	Filter the result down to a limited set of IDs by using event=id1;id2.

1.19.3.1. Examples

Query for all events with children of a certain organisation unit:

```
/api/25/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/25/events.json?orgUnit=O6uvpzGd5pu&ouMode=DESCENDANTS
```

Query for all events with a certain program and organisation unit:

```
/api/25/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc
```

Query for all events with a certain program and organisation unit, sorting by due date ascending:

```
/api/25/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/25/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/25/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/25/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/25/events.json?orgUnit=DiszpKrYNg8&program=eBAyeGv0exc
  &trackedEntityInstance=gfVxE3ALA9m&startDate=2014-01-01&endDate=2014-12-31
```

1.20. Program rules

This section is about sending and reading program rules, and explains the program rules data model. The program rules gives functionality to configure dynamic behavior in the programs in DHIS.

1.20.1. Program rules 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.

DHIS2 Login screen
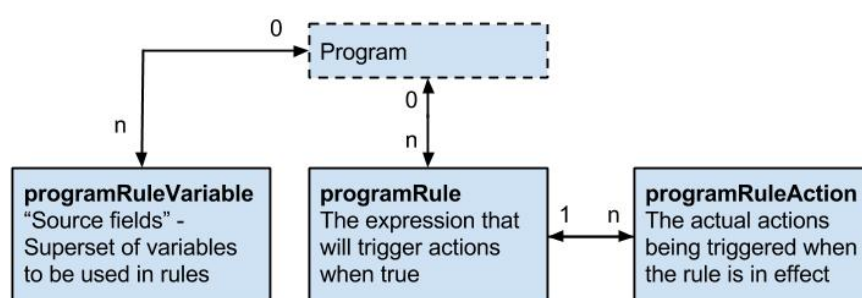
1.20.1.1. Program rule model details

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

Table 1.36. programRule

name	description	Compulsory
program	The program of which the programRule is executed in.	Compulsory

name	description	Compulsory
name	The name with which the program rule will be displayed to dhis configurators. Not visisble 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 visisble 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

1.20.1.2. Program rule action model details

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

Table 1.37. 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.	Compulsory

name	description	Compulsory
	• 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. • trackedEntityAttribute - if defined*, the warning message is displayed next to this tracked entity attribute. *Either dataElement or trackedEntityAttribute must be specified. • SHOWERROR - Show an error to the user, blocking the user from completing the event or registration. • content - if defined, content is a static part that is displayed in the start of the error message. • data - if defined, data forms an expression that is evaluated and added to the end of the error message. • dataElement - if defined*, the error message is linked to this data element. • trackedEntityAttribute - if defined*, the error message is linked to this tracked entity attribute. *Either dataElement or trackedEntityAttribute must be specified. • WARNINGONCOMPLETINON - Show a warning to the user on the "Complete form" dialog, but allowing the user to complete the event. • 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 prefixed with the name/formName of the data element.	

name	description	Compulsory

	• ERRORONCOMPLETION - Show an error to the user on in a modal window when the user tries to complete the event. The user is prevented from completing the event.
	
	  • content - if defined, content is a static part that is displayed in the start of the error message.
	
	  • data - if defined, data forms an expression that is evaluated and added to the end of the error message.
	
	  • dataElement - if defined, the error message is linked to this data element.
	
	• CREATEEVENT - Create an event within the same enrollment.
	
	  • content
	
	  • data - if defined, contains data values to assign the created event. The format is <uid>:<data value>. Where several values is specified, these are separated with comma.
	
	    `AcMrnleqHqc:100,AqK1IHqCkEE:'Polyhydramnios'`
	
	  • programStage - must be defined, and designates the program stage that the rule shall create an event of.	

location	Used for actionType DISPLAYKEYVALUEPAIR and DISPLAYTEXT to designate which widget to display the text or keyvalyepair 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
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

1.20.1.3. Program rule variable model details

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

Table 1.38. programRuleVariable

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

1.20.2. Posting program rules through the API

-coming-

1.21. 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 label and identifiers. By supplying period and organisation unit identifiers the form response will be populated with data values.

Table 1.39. 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/25/dataSets/<dataset-id>/form.json
```

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

```
/api/25/dataSets/BfMAe6Itzgt/form
```

To retrieve the form including metadata in JSON:

```
/api/25/dataSets/BfMAe6Itzgt/form.json?metaData=true
```

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

```
/api/25/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/25/dataSets/BfMAe6Itzgt/form"
  -H "Content-Type:text/html" -u admin:district -X PUT -v
```

1.22. 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/25/validation/dataSet/QX4ZTUbOt3a.json?pe=201501&ou=DiszpKrYNg8
```

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 to validation rules. This will be extended with more validation types in 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/25/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.

Table 1.40. 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

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.

Table 1.41. 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

1.23. 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.

1.23.1. 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://dhis.domain/api/25/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. 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.

1.23.2. Fetching the result

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

```
curl -X GET https://dhis.domain/api/25/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.

1.24. Indicators

This section describes indicators and indicator expressions.

1.24.1. Aggregate indicators

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

```
/api/25/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, 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.

Table 1.42. Indicator variables

Variable	Object	Description
#{<dataelement-id>.<catoptcombo-id>}	Data element operand	Refers to a combination of an aggregate data element and a category option combination.
#{<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.
I{program-indicator-id>	Program indicator	Refers to the value of a program indicator.
C{<constant-id>}	Constant	Refers to a constant value.
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
```

An example which use 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}
```

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

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

1.24.2. Program indicators

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

```
/api/25/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.

Table 1.43. Program indicator variables

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

1.24.3. 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/25/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"
}
```

1.25. 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.

1.25.1. Completing and un-completing data sets

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

```
/api/25/completeDataSetRegistrations
```

This resource supports the methods POST for registration and DELETE for un-registration. The following query parameters are supported:

Table 1.44. 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)
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

1.25.2. 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:

Table 1.45. Data value set query parameters

Parameter	Description
dataSet	Data set identifier, can be specified multiple times
period	PeriodType
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 specified multiple times
children	Whether to include the children in the hierarchy of the organisation units

The dataSet and orgUnit parameters can be repeated in order to include multiple data sets and organisation units. An example request looks like this:

```
curl "https://play.dhis2.org/demo/api/25/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 -v
```

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.

1.26. Data approval

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

```
/api/25/dataApprovals
```

1.26.1. Get approval status

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

```
GET http://server.com/api/dataApprovals?ds=aLpVgfXiz0f&pe=2013&ou=DiszpKrYNg8
```

Table 1.46. Data approval query parameters

Query parameter	Required	Description
ds	Yes	Data set identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier

This will give you a response something like this:

```
{
  "mayApprove": false,
  "mayUnapprove": false,
  "mayAccept": false,
  "mayUnaccept": false,
  "state": "UNAPPROVED_ELSEWHERE"
}
```

The returned parameters are:

Table 1.47. Data approval query 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.

Table 1.48. 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 approvede. 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:

```
GET api/dataApprovals/categoryOptionCombos?ds=aLpVgfXiz0f&pe=2013&ou=DiszpKrYNg8
```

1.26.2. 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/25/dataApprovals
```

To accept data you can issue a POST request to the dataApprovals/acceptances resource. To un-accept data you can issue a DELETE request to the dataApprovals/acceptances resource.

```
POST DELETE /api/25/dataApprovals/acceptances
```

These requests contain the following parameters:

Table 1.49. Data approval action parameters

Action parameter	Required	Description
ds	Yes	Data set identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit 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, all of the following must be true:

• The organisation unit's level must be specified by an approval level.

• The time period specified must match the period type of the data set.

• The data set must specify that data can be approved for this data set.

1.26.3. Bulk approve data

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

```
POST /api/25/dataApprovals/approvals
```

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

```
POST /api/25/dataApprovals/unapprovals
```

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

```
POST /api/25/dataAcceptances/acceptances
```

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

```
POST /api/25/dataAcceptances/unacceptances
```

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

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

1.27. Auditing

DHIS2 does automatic auditing on all update and deletions of aggregate data values, tracked entity data values, tracked entity attribute values, and this chapter will explain the web-api available for fetching this data back out again.

1.27.1. 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.

Table 1.50. 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/25/audits/dataValue?ds=lyLU2wR22tC
```

1.27.2. 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.

Table 1.51. 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/25/audits/trackedEntityDataValue?de=eMyVanycQSC&de=qrur9Dvnyt5
```

1.27.3. 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.

Table 1.52. 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/25/audits/trackedEntityAttributeValue?tea=VqEFza8wbwA
```

1.28. 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/25/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.

1.28.1. 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 http://play.dhis2.org/demo/api/24/users. We continue by following link to the mobile user at http://play.dhis2.org/demo/api/24/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 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/25/messageConversations"
  -H "Content-Type:application/xml" -u admin:district -X POST -v
```

A corresponding payload in JSON and POST command look 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/25/messageConversations"
  -H "Content-Type:application/json" -u admin:district -X POST -v
```

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 te 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/25/messageConversations"
  -H "Accept:application/xml" -u mobile:district -X GET -v
```

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. From the description at http://play.dhis2.org/demo/api/24/messageConversations we learned that we can reply directly to an existing message conversation once we know the URL by including the message text as the request payload (body). 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/25/messageConversations/ZjHHSjyyeJ2"
  -H "Content-Type:text/plain" -u mobile:district -X POST -v
```

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

1.28.2. Managing messages

Note: the Web-API calls discussed in this section were introduced in DHIS 2.17

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 users message 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 not 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/25/messageConversations/jMe43
```

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/25/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/25/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 errorenous 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 -v
```

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

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

1.28.3. Tickets

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 do this, use these API

```
POST https://play.dhis2.org/demo/api/25/messageConversations/<uid>/status
```

```
POST https://play.dhis2.org/demo/api/25/messageConversations/<uid>/priority
```

Table 1.53. 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/25/messageConversations/ZjHHSjyyeJ2?internal=true"
  -H "Content-Type:text/plain" -u admin:district -X POST -v
```

1.29. Interpretations

For resources related to data analysis in DHIS 2, 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/25/interpretations
```

1.29.1. Reading interpretations

To read interpretations we will interact with the /api/25/interpretations resource. A typical GET request using field filtering can look like this:

```
GET /api/25/interpretations?fields=*,comments[id,text,user]
```

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"
        }
    }, {
        "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"
        },
        "comments": [{
```

```
            "id": "iB4Etq8yTE6",
            "text": "This report indicates a surge",
            "user": {
                "id": "B4XIfwOcGyI"
            }, {
            "id": "iB4Etq8yTE6",
            "text": "Likely caused by heavy rainfall",
            "user": {
                "id": "B4XIfwOcGyI"
            }]
        }
    }]
}
```

Table 1.54. Interpretation fields

Field	Description
id	The interpretation identifier.
created	The time of when the interpretation was created.
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 creating the interpretation.
reportTable	Association to the report table if type is REPORT_TABLE.
chart	Association to the chart if type is CHART.
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.

For all analytical objects you can append /data to the URL to retrieve the data associated with the resource (as apposed 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/25/maps/bhmHJ4ZCdCd/data
```

1.29.2. 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 obejct being interpreted.

```
/api/25/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.

```
/api/25/interpretations/reportTable/yC86zJxU1i1
/api/25/interpretations/chart/ZMuYVhtIceD
/api/25/interpretations/map/FwLHSMCejFu
```

```
/api/25/interpretations/eventReport/xJmPLGP3Cde
/api/25/interpretations/eventChart/nEzXB2M9YBz
/api/25/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/25/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/25/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.

1.29.3. 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/25/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/25/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.

1.29.4. 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/25/interpretations/{interpretation-id}/comment
```

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/25/ interpretations/{id}/comment" resource like this:

```
curl -d "An intervention is needed" -X POST
  "https://play.dhis2.org/demo/api/25/interpretations/j8sjHLkK8uY/comment"
  -H "Content-Type:text/plain" -u admin:district -v
```

1.29.5. 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/25/interpretations/{interpretation-id}/comments/{comment-id}
```

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

```
curl -d "I agree with that." -X PUT
  https://play.dhis2.org/demo/api/25/interpretations/j8sjHLkK8uY/comment/idAzzhVWvh2"
  -H "Content-Type:text/plain" -u admin:district -v
```

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

1.29.6. Liking interpretations

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

```
POST /api/25/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 intepretation can be viewed by looking at the regular Web API representation:

```
GET /api/25/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"
    }]
}
```

1.30. 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 modication 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.

Table 1.55. 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

Resource	Description	Data URL	Resource representations
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.

Table 1.56. Data query parameters

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

Table 1.57. 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/25/charts/R0DVGvXDUNP/data
/api/25/charts/R0DVGvXDUNP/data?date=2013-06-01

/api/25/reportTables/jIISuEWxmoI/data.html
/api/25/reportTables/jIISuEWxmoI/data.html?date=2013-01-01
/api/25/reportTables/FPmvWs7bn2P/data.xls
/api/25/reportTables/FPmvWs7bn2P/data.pdf

/api/25/maps/DHE98Gsynpr/data
/api/25/maps/DHE98Gsynpr/data?date=2013-07-01

/api/25/reports/OeJsA6K1Otx/data.pdf
/api/25/reports/OeJsA6K1Otx/data.pdf?date=2014-01-01
```

1.31. Plugins

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

1.31.1. 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 Ext JS 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>
```

```html
<head>
  <link rel="stylesheet" type="text/css" href="http://dhis2-cdn.org/v215/ext/
resources/css/ext-plugin-gray.css" />
  <script src="https://dhis2-cdn.org/v215/ext/ext-all.js"></script>
  <script src="https://dhis2-cdn.org/v215/plugin/table.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() {

      // Referring to an existing table through the id parameter, render to "table1"
 div

      DHIS.getTable({ url: base, el: "table1", id: "R0DVGvXDUNP" });

      // Full table configuration, render to "table2" div

      DHIS.getTable({
        url: base,
        el: "table2",
        columns: [
          {dimension: "de", items: [{id: "YtbsuPPo010"}, {id: "l6byfWFUGaP"}]}
        ],
        rows: [
          {dimension: "pe", items: [{id: "LAST_12_MONTHS"}]}
        ],
        filters: [
          {dimension: "ou", items: [{id: "USER_ORGUNIT"}]}
        ],
        // All following options are optional
        showTotals: false,
        showSubTotals: false,
        hideEmptyRows: true,
        showHierarchy: true,
        displayDensity: "comfortable",
        fontSize: "large",
        digitGroupSeparator: "comma",
        legendSet: {id: "BtxOoQuLyg1"}
      });
    }
  </script>
</head>

<body>
  <div id="table1"></div>
  <div id="table2"></div>
</body>
</html>
```

Three files 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

css stylesheet. The third file is the Pivot table 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="http://dhis2-cdn.org/v215/plugin/table.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.

```
var base = "https://play.dhis2.org/demo/";

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 Pivot table plug-in. Two properies are
required: el and url (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 shoud 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: 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 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: "in", // "in", "de", "ds", "dc", "pe", "ou" or any dimension id
  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
  ]
}]
```

Table 1.58. Pivot table plug-in configuration

Param	Type	Required	Options (default first)	Description
el	string	Yes		Identifier of the HTML element to render the table in your web page
url	string	Yes		Base URL of the DHIS2 server

Param	Type	Required	Options (default first)	Description
id	string	No		Identifier of a pre-defined table (favorite) in DHIS
columns	array	Yes (if no id provided)		Data dimensions to include in table as columns
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
showTotals	boolean	No	true | false	Whether to display totals for columns and rows
showSubTotals	boolean	No	true | false	Whether to display sub-totals for columns and rows
hideEmptyRows	boolean	No	false | true	Whether to hide rows with no data
showHierarchy	boolean	No	false | true	Whether to extend orgunit names with the name of all anchestors
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		Show a color indicator next to the values (currently reusing legend sets from GIS)

Web API
Embedding charts with the
Visualizer chart plug-in

Param	Type	Required	Options (default first)	Description
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with curretn user, single or array

We continue by adding one pre-defined and one dynamic pivot table to our HTML document. You can browse the list of available pivot tables using the Web API here: http://play.dhis2.org/demo/api/24/reportTables.

```
function setLinks() {
  DHIS.getTable({ url: base, el: "table1", id: "R0DVGvXDUNP" });

  DHIS.getTable({
url: base,
el: "table2",
columns: [
   {dimension: "de", items: [{id: "YtbsuPPo010"}, {id: "l6byfWFUGaP"}]}
],
rows: [
   {dimension: "pe", items: [{id: "LAST_12_MONTHS"}]}
],
filters: [
   {dimension: "ou", items: [{id: "USER_ORGUNIT"}]}
],
// All following options are optional
showTotals: false,
showSubTotals: false,
hideEmptyRows: true,
showHierarchy: true,
displayDensity: "comfortable",
fontSize: "large",
digitGroupSeparator: "comma",
legendSet: {id: "BtxOoQuLyg1"}
  });
}
```

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="table1"></div>
<div id="table2"></div>
```

To see a complete working example please visit http://play.dhis2.org/portal/table.html.

1.31.2. Embedding charts with the Visualizer chart plug-in

In this example we will see how we can embed good-looking Ext JS charts (http://www.sencha.com/products/extjs) 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 Ext JS library only. 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.

Web API
Embedding charts with the
Visualizer chart plug-in

We start by having a look at what the complete html file could look like. This setup puts two charts in our web page. The first one is referring to an existing chart. 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="http://dhis2-cdn.org/v215/plugin/chart.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() {

      // Referring to an existing chart through the id parameter, render to "chart1"
 div

      DHIS.getChart({ url: base, el: "chart1", id: "R0DVGvXDUNP" });

      // Full chart configuration, render to "chart2" div

      DHIS.getChart({
        url: base,
        el: "chart2",
        type: "stackedBar",
        columns: [ // Chart series
          {dimension: "de", items: [{id: "YtbsuPPo010"}, {id: "l6byfWFUGaP"}]}
        ],
        rows: [ // Chart categories
          {dimension: "pe", items: [{id: "LAST_12_MONTHS"}]}
        ],
        filters: [
          {dimension: "ou", items: [{id: "USER_ORGUNIT"}]}
        ],
        // All following options are optional
        showData: false,
        targetLineValue: 70,
        baseLineValue: 20,
        showTrendLine: true,
        hideLegend: true,
        title: "My chart title",
        domainAxisTitle: "Periods",
        rangeAxisTitle: "Percent"
      });
    }
  </script>
</head>

<body>
```

```
    <div id="chart1"></div>
    <div id="chart2"></div>
</body>
</html>
```

Three files 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 Visualizer 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="http://dhis2-cdn.org/v215/plugin/chart.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.

```
var base = "https://play.dhis2.org/demo/";

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 Visualizer plug-in. Two properies are required: el and url (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 shoud omit the id parameter and provide data dimensions inside a columns array (chart series), a rows array (chart categories) 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 e.g. "ANC 1 Coverage", "ANC 2 Coverage" and "ANC 3 Coverage" as series in your chart 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
    {id: "OdiHJayrsKo"}, // the id of ANC 2 Coverage
    {id: "sB79w2hiLp8"}  // the id of ANC 3 Coverage
  ]
}]
```

Table 1.59. Visualizer chart plug-in configuration

Param	Type	Required	Options (default first)	Description
el	string	Yes		Identifier of the HTML element to render the

Param	Type	Required	Options (default first)	Description
				chart in your web page
url	string	Yes		Base URL of the DHIS2 server
id	string	No		Identifier of a pre-defined chart (favorite) in DHIS
type	string	No	column | stackedcolumn | bar | stackedbar | line | area | pie	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
showData	boolean	No	false | true	Whether to display data on the chart
showTrendLine	boolean	No	false | true	Whether to display trend line(s) on the chart
hideLegend	boolean	No	false | true	Whether to hide the chart legend
hideTitle	boolean	No	false | true	Whether to hide the chart title
targetLineValue	double	No		Value of target line to display on the chart
targetLineLabel	string	No		Label for target line
baseLineValue	double	No		Value of baseline to display on the chart
baseLineLabel	string	No		Label for baseline
domainAxisTitle	string	No		Title for the domain axis

Param	Type	Required	Options (default first)	Description
rangeAxisTitle	string	No		Title for the range axis
width	integer	No		Width of chart
height	integer	No		Height of chart
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with curretn user, single or array

We continue by including two pre-defined charts and to dynamic charts to our HTML document. You can browse the list of available charts using the Web API here: http://play.dhis2.org/demo/api/24/charts.

```
function setLinks() {
  DHIS.getChart({ url: base, el: "chart1", id: "R0DVGvXDUNP" });

  DHIS.getChart({
url: base,
el: "chart2",
type: "stackedBar",
columns: [ // Chart series
  {dimension: "de", items: [{id: "YtbsuPPo010"}, {id: "l6byfWFUGaP"}]}
],
rows: [ // Chart categories
  {dimension: "pe", items: [{id: "LAST_12_MONTHS"}]}
],
filters: [
  {dimension: "ou", items: [{id: "USER_ORGUNIT"}]}
],
// All following options are optional
showData: false,
targetLineValue: 70,
baseLineValue: 20,
showTrendLine: true,
hideLegend: true,
title: "My chart title",
domainAxisTitle: "Periods",
rangeAxisTitle: "Percent"
  });
}
```

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="chart1"></div>
<div id="chart2"></div>
```

To see a complete working example please visit http://play.dhis2.org/portal/chart.html.

1.31.3. 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 in our web page. The first one is referring to an existing map. The second is configured inline.

```html
<!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 properies 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 shoud 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
}]
```

Table 1.60. 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

Param	Type	Required	Options (default first)	Description
				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:

Table 1.61. 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 curretn 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: <u>http://</u> <u>play.dhis2.org/demo/api/24/maps</u>.

```
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.

1.31.4. Creating a chart carousel with the carousel plug-in

The chart plug-in also makes it possible to create a chart carousel which for instance can be used to create an attractive front page on a Web portal. To use the carousel we need to import a few files in the head section of our HTML page:

```
<link rel="stylesheet" type="text/css" href="http://dhis2-cdn.org/v213/ext/resources/
css/ext-plugin-gray.css" />
<link rel="stylesheet" type="text/css" href="https://play.dhis2.org/demo/dhis-web-
commons/javascripts/ext-ux/carousel/css/carousel.css" />
<script type="text/javascript" src="https://extjs-public.googlecode.com/svn/tags/
extjs-4.0.7/release/ext-all.js"></script>
<script type="text/javascript" src="https://play.dhis2.org/demo/dhis-web-commons/
javascripts/ext-ux/carousel/Carousel.js"></script>
<script type="text/javascript" src="https://play.dhis2.org/demo/dhis-web-commons/
javascripts/plugin/plugin.js"></script>
```

The first file is the CSS stylesheet for the chart plug-in. The second file is the CSS stylesheet for the carousel widget. The third file is the Ext JavaScript framework which this plug-in depends on. The fourth file is the carousel plug-in JavaScript file. The fifth file is the chart plug-in JavaScript file. The paths in this example points at the DHIS2 demo site, make sure you update them to point to your own DHIS2 installation.

Please refer to the section about the chart plug-in on how to do authentication.

To create a chart carousel we will first render the charts we want to include in the carousel using the method described in the chart plug-in section. Then we create the chart carousel itself. The charts will be rendered into div elements which all have a CSS class called chart. In the carousel configuration we can then define a selector expression which refers to those div elements like this:

```
DHIS.getChart({ uid: 'R0DVGvXDUNP', el: 'chartA1', url: base });
DHIS.getChart({ uid: 'X0CPnV6uLjR', el: 'chartA2', url: base });
DHIS.getChart({ uid: 'j1gNXBgwKVm', el: 'chartA3', url: base });
DHIS.getChart({ uid: 'X7PqaXfevnL', el: 'chartA4', url: base });
```

```
new Ext.ux.carousel.Carousel( 'chartCarousel', {
  autoPlay: true,
  itemSelector: 'div.chart',
  interval: 5,
  showPlayButton: true
});
```

The first argument in the configuration is the id of the div element in which you want to render the carousel. The autoPlay configuration option refers to whether we want the carousel to start when the user loads the Web page. The interval option defines how many seconds each chart should be displayed. The showPlayButton defines whether we want to render a button for the user to start and stop the carousel. Finally we need to insert the div elements in the body of the HTML document:

```
<div id="chartCarousel">

<div id="chartA1"></div>
<div id="chartA2"></div>
<div id="chartA3"></div>
<div id="chartA4"></div>
```

To see a complete working example please visit http://play.dhis2.org/portal/carousel.html.

1.32. 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/25/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 DHIS 2 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/25/sqlViews/dI68mLkP1wN/execute" -X POST -u
 admin:district -v
```

The next step in the process is the retrieval of the data.The basic structure of the URL is as follows

```
http://{server}/api/25/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/25/sqlViews/dI68mLkP1wN/data.csv" -u
 admin:district -v
```

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.

1.32.1. 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/25/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/25/sqlViews/dI68mLkP1wN/data.csv?criteria=level:4
```

1.32.2. Variables

SQL views support variable subsitution. Variable subsitition 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/25/sqlViews/dI68mLkP1wN/data.json?var=valueType:int
```

The valueType variable will be subsituted 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.

1.33. 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/25/dashboards
```

1.33.1. 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/25/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/25/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/25/dashboards/vQFhmLJU5sK/?fields=:all,dashboardItems[:all]
```

1.33.2. Searching dashboards

When setting a dashboard it is convenient from a consumer point of view 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/25/dashboards/q/my-query.json
```

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"
    }],
    "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
}
```

1.33.3. 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/dasboards resource.

To remove a dashboard, you can make a DELETE request to the specific dashboard resource similar to this:

```
/api/25/dashboards/vQFhmLJU5sK
```

1.33.4. 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.

Table 1.62. Items content parameters

Query parameter	Description	Options
type	Type of the resource to be represented by the dashboard item	chart | 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/25/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/25/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/25/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 dasboard 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/25/dashboards/<dashboard-id>/items/<item-id>/content/<content-resource-id>
```

1.34. 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/25/analytics
```

1.34.1. Request query parameters

The analytics resource lets you specify a range of query parameters:

Table 1.63. Query parameters

Query parameter	Required	Description	Options
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 | COUNT | STDDEV | VARIANCE
measureCriteria	No	Filters for the data/measures.	EQ | GT | GE | LT | LE
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 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 response.	false | true

Query parameter	Required	Description	Options
hideEmptyRows	No	Hides empty rows 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
displayProperty	No	Property to display for metadata.	NAME | SHORTNAME
outputIdScheme	No	Property to use for metadata items the query response, can be identifier, code or name.	UID | CODE | NAME
inputIdScheme	No	Property to use for metadata items in the request, can be identifier or code.	UID | CODE
approvalLevel	No	Include data which has been approved at least up to the given approval level, refers to identfier 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 current user, multiple identifiers can be separated by semi-colon.	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)

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/25/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/25/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 critieria and value should be substituted with real values:

```
/api/25/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/25/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&dimension=pe:2014
  &dimension=ou:O6uvpzGd5pu;lc3eMKXaEfw&measureCriteria=GE:6500;LT:33000
```

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 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/25/analytics.html?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU&dimension=pe:2014Q1;2014Q2
  &dimension=ou:O6uvpzGd5pu&tableLayout=true&columns=dx;ou&rows=pe
```

1.34.2. 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:

Table 1.64. 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	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.
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>

Dimension	Dimension id	Dimension items
Category option combinations	co	Category option combo identifers (omit to get all items)
Attribute option combinations	ao	Category option combo identifers (omit to get all 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/25/dimensions
```

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/25/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/25/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/25/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-<id> syntax:

```
/api/25/analytics?dimension=dx:DE_GROUP-h9cuJOkOwY2
  &dimension=pe:201601&dimension=ou:O6uvpzGd5pu
```

When selecting data set reporting rates, the syntax contains of a data set identifier followed by a reporting rate metric:

```
/api/25/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 <program-id>.<dataelement-id> syntax:

```
/api/25/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 <program.id>.<attribute-id> syntax:

```
/api/25/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 dimension identifier and the groups as dimension items:

```
/api/25/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 dimension identifier and the category options as dimension items:

```
/api/25/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/25/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. A simple query for all org units at level three:

```
/api/25/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/25/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-<id> 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/25/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2016
  &dimension=ou:OU_GROUP-w0gFTTmsUcF;OU_GROUP-EYbopBOJWsW;O6uvpzGd5pu;lc3eMKXaEf
```

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 they 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.

1.34.3. The dx dimension

The dx dimension is a special dimension which can contain all of the following data types.

Table 1.65. Data dx dimension types

Type	Syntax	Description	Data source
Indicator	<indicator-id>	Indicator identifier.	Aggregated data
Indicator	IN_GROUP-<indicatorgroup-id>	Keyword followed by 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	DE_GROUP-<dataelementgroup-id>	Keyword followed by data element group identifier. Will include all data elements in the group in the response.	Aggregated 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.	Aggregated data
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

Items from all of the various dx types can be combined in an analytics request. An example looks like this:

```
/api/25/analytics.json
```

```
  ?
dimension=dx:Uvn6LCg7dVU;BfMAe6Itzgt.REPORTING_RATE;BfMAe6Itzgt.ACTUAL_REPORTS;IpHINAT79UW.a3kGc0
  &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/25/analytics.json
  ?dimension=dx:DE_GROUP-qfxEYY9xAl6;IN_GROUP-oehv9EO3vP7;BfMAe6Itzgt.REPORTING_RATE
  &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.

1.34.4. 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
- xls (application/vnd.ms-excel)

As an example, to request an analytics response in XML format you can use the following URL:

```
/api/25/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/25/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 the 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 aggregeted data values.

1.34.5. Constraints

There are several constraints on the input you can provide to the analytics resource.
- At least one dimension must be specified in a query.
- Dimensions cannot be specified as dimension and filter simultaneously.
- At least one period must be specified as dimension or filter.
- Categories cannot be specified as filter.
- Only a single indicator can be specified as filter.
- Only a single reporting rate can be specified as filter.
- Data element group sets cannot be specified together with data sets.
- Categories can only be specified together with data elements, not indicators or data sets.
- A dimension cannot be specified more than once.
- Fixed dimensions ("dx", "pe", "ou") must have at least one option if included in a query.
- A table cannot contain more than 50 000 cells by default, this can be configured under system settings.

When a query request violates any of these constraints the server will return a response with status code 409 and content-type "text/plain" together with a textual description of the problem.

1.34.6. Data value set format

The analytics 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 destionation 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 analytics/dataValueSethttp://dhis2.github.io/dhis2-docs/master/en/developer/html/webapi_authentication.html#webapi_oauth2http://dhis2.github.io/dhis2-docs/master/en/developer/html/webapi_authentication.html#webapi_oauth2 resource:

```
/api/25/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 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
```

1.34.7. 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 is identical to regular analytics queries:

```
/api/25/analytics/debug/sql?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
   &filter=pe:2016Q1;2016Q2&filter=ou:O6uvpzGd5pu
```

1.35. 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/25/analytics/events
```

1.35.1. 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.

Table 1.66. 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>

Dimension	Dimension id	Description
Organisation unit group sets	<org unit group set id>	Organisation unit group set identifiers
Categories	<category id>	Category identifiers (program attribute categories only)

1.35.2. Request query parameters

The analytics event API let you specify a range of query parameters.

Table 1.67. Query parameters for both event query and aggregate analytics

Query parameter	Required	Description	Options
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
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

Table 1.68. 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	DESCENDANTS, CHILDREN, SELECTED

Query parameter	Required	Description	Options
		hierarchy; SELECTED refers to the selected organisation units only.	
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
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

Table 1.69. 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.	AVERAGE | SUM | 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
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

Query parameter	Required	Description	Options
aggregateData	No	Produce aggregate values for the data dimensions (as opposed to dimension items).	false | true

Table 1.70. Query parameters for cluster event analytics only

Query parameter	Required	Description	Options
clusterSize	Yes	Size of clusters in meters.	Numeric positive value
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

1.35.3. 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/25/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/25/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/25/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/25/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/25/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/25/analytics/events/query/eBAyeGv0exc?startDate=2016-01-01&endDate=2016-10-31
  &dimension=ou:O6uvpzGd5pu&dimension=qrur9Dvnyt5&page=3&pageSize=20
```

1.35.3.1. 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.

Table 1.71. 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 ";"

1.35.3.2. 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/25/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/25/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 organistion 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.

1.35.4. 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/25/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/25/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/25/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/25/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/25/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/25/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/25/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/25/analytics/events/aggregate/eBAyeGv0exc.json?
stage=Zj7UnCAulEk&dimension=ou:ImspTQPwCqd

 &dimension=pe:LAST_12_MONTHS&dimension=fWIAEtYVEGk&value=qrur9Dvnyt5&aggregationType=AVERAGE
```

1.35.4.1. 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/25/analytics/events/aggregate/eBAyeGv0exc.json?
stage=Zj7UnCAulEk&dimension=qrur9Dvnyt5-Yf6UHoPkdS6
  &dimension=ou:ImspTQPwCqd&dimension=pe:LAST_12_MONTHS
```

1.35.4.2. 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:

```json
{
    "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.

1.35.5. Event clustering analytics

The analytics/events/cluster resource provides clustered geospatial event data. A request looks like this:

```
/api/25/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 centre 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)",
""]
    ]
}
```

1.35.6. 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/25/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
}
```

1.36. Data set report

Data set reports can be generated trough the web api using the /dataSetReport resource. This resource generates reports on data set and returns the result in the form of a HTML table.

```
/api/25/dataSetReport
```

The request supports the following parameters:

Table 1.72. Accepted parameters of /dataSetReport resource

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

Parameter	Description	Type	Required
dimension	Dimensions to be used as filters for the report	One or more UIDs	No
selectedUnitOnly	Whether to use captured or aggregated data	Boolean	No

The data set report resource accepts GET requests only. An example request to retrieve a report for a data set and orgunit for 2015 looks like this:

```
GET /api/dataSetReport?ds=BfMAe6Itzgt&pe=201610&ou=ImspTQPwCqd&selectedUnitOnly=false
```

1.37. Push Analysis

The push analysis api includes endpoints for previewing a push analysis report for the logged in user and manualy 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 pushanalysis to be disabled, the job will also be stopped from running in the future.

To get a HTML preview of an existing push analysis, you can do a GET request to the following endpoint:

```
/api/25/pushAnalysis/<push-analysis-uid>/render
```

To manually trigger a push analysis job, you can do a GET request to this endpoint:

```
/api/25/pushAnalysis/<push-analysis-uid>/run
```

A push analysis consists of the following properties, where some are required to automaticly run push analysis jobs:

Table 1.73. 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	No. Push analysis without a valid day of frequency for the frequency set will not be scheduled.

Property	Description	Type	Required
		when frequency is "MONTHLY"	

1.38. 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/25/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.

1.38.1. Request query parameter

The usage analytics (data statistics) API supports two operations:
- POST: creates a view event
- GET: retrieves aggregated statistics

1.38.2. 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/dataStatistics?eventType=CHART_VIEW&favorite=LW0O27b7TdD
```

A successful save operation returns HTTP status code 201. The table below shows the supported types of events.

Table 1.74. 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

1.38.3. Retrieve aggregated usage analytics report (GET)

The usage analytics (data statistics) API lets you specify certain query parameters when asking for an aggregated report.

Table 1.75. 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 yearly aggregation:

```
GET /api/dataStatistics?startDate=2014-01-02&endDate=2016-01-01&interval=MONTH
```

1.38.4. Retrieve top favourites

The usage analytics API lets you retrieve the top favorites used in DHIS2, and by user.

Table 1.76. 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 username, and will then find the top favorites of the system. If username is specified, the response will only contain the top favorites of that user.

```
/api/dataStatistics/favorites?eventType=CHART_VIEW&pageSize=25&sortOrder=ASC
```

```
/api/dataStatistics/favorites?
eventType=CHART_VIEW&pageSize=25&sortOrder=ASC&username=admin
```

1.38.5. 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 an usage analytics response in XML format:

```
/api/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.

Execute this command against the demo database to get an usage analytics response in JSON format:

```
curl "play.dhis2.org/demo/api/dataStatistics?startDate=2016-02-01&endDate=2016-02-14&
interval=WEEK" -u admin:district
```

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
  }
]
```

1.39. 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 reading 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/25/geoFeatures.json?ou=ou:LEVEL-3
```

To retrieve geo features for organisation units at level within the boundary of an organisation unit (e.g. at level 2) you can use this URL:

```
/api/25/geoFeatures.json?ou=ou:LEVEL-4;O6uvpzGd5pu
```

The semantics of the response properties are described in the following table.

Table 1.77. 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

1.39.1. GeoJSON

To export GeoJSON, you can simple 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 (defaults to 1) and parent (defaults to root organisation units), both can be added multiple times, some examples follow.

Get all features at level 2 and 4:

```
/api/25/organisationUnits.geojson?level=2&level=4
```

Get all features at level 3 with a boundary organisation unit:

```
/api/25/organisationUnits.geojson?parent=fdc6uOvgoji&level=3
```

1.40. Generating resource and analytics tables

DHIS2 features a set of generated database tables which are used as 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/25/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/25/resourceTables/analytics
```

Table 1.78. 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
lastYears	integer	Number of last years of data to include

These requests will return immediately and initiate a server-side process.

1.41. Maintenance

To perform maintenance you can interact with the maintenance resource. You should use POST or PUT as method for requests. The following requests are available.

Analytics tables clear will drop all analytics tables:

```
/api/25/maintenance/analyticsTablesClear
```

Expired invitations clear will remove all user account invitations which have expired:

```
/api/25/maintenance/expiredInvitationsClear
```

Period pruning will remove periods which are not linked to any data values:

```
/api/25/maintenance/periodPruning
```

Zero data value removal will delete zero data values linked to data elements where zero data is defined as not significant:

```
/api/25/maintenance/zeroDataValueRemoval
```

Drop SQL views will drop all SQL views in the database. Note that it will not delete the DHIS2 SQL views.

```
/api/25/maintenance/sqlViewsDrop
```

Create SQL views will recreate all SQL views in the database.

```
/api/25/maintenance/sqlViewsCreate
```

Category option combo update will remove obsolete and generate missing category option combos for all category combinations:

```
/api/25/maintenance/categoryOptionComboUpdate
```

Cache clearing will clear the application Hibernate cache and the analytics partition caches:

```
/api/25/maintenance/cacheClear
```

Re-generate organisation unit path property (can be useful if you imported org units with SQL):

```
/api/25/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.

```
/api/25/maintenance/dataPruning/organisationUnits/<org-unit-id>
```

Metadata validation will apply all metadata validation rules and return the result of the operation:

```
/api/25/metadataValidation
```

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:

```
/api/25/maintenance?
analyticsTablesClear=true&expiredInvitationsClear=true&periodPruning=true

 &zeroDataValueRemoval=true&sqlViewsDrop=true&sqlViewsCreate=true&categoryOptionComboUpdate=true
  &cacheClear=true&ouPathsUpdate=true
```

1.42. System resource

The system resource provides you with convenient information and functions. The system resource can be found at /api/system.

1.42.1. Generate identifiers

To generate valid, random DHIS2 identifiers you can do a GET request to this resource:

```
/api/25/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 a 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).

1.42.2. View system information

To get information about the current system you can do a GET request to this URL:

```
/api/25/system/info
```

JSON and JSONP response formats are supported. The system info response currently includes the below properties. Note that if the user who is requesting this resourec does not have full authority in the system then only the first seven properties will be included, as this information is security sensitive.

```
{
    contextPath: "http://yourdomain.com",
    userAgent: "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 Chrome/29.0.1547.62
Safari/537.36",
    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
}
```

To get information about the system context (contextPath and userAgent) only you can do a GET request to the below URL. JSON and JSONP response formats are supported:

```
/api/25/system/context
```

1.42.3. 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/25/system/ping
```

You can detect the outcome of the authentication by inspecting the HTTP status code of the response header. The meaning of the possible status codes are listed below. Note that this applies to Web API requests in general.

Table 1.79. HTTP Status codes

HTTP Status code	Description	Outcome
200	OK	Authentication was successful
302	Found	No credentials was supplied with the request - no authentication took place
401	Unauthorized	The username and password combination was incorrect - authentication failed

1.42.4. View asynchronous task status

Several tasks which typically take a significant time to complete can be performed asynchronously. After initiating an async task you can poll the status through the system/tasks resource by suppling the task category 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.

Table 1.80. Task categories

Identifier	Description
ANALYTICSTABLE_UPDATE	Generation of the analytics tables.
RESOURCETABLE_UPDATE	Generation of the resource tables.
MONITORING	Processing of data surveillance/monitoring validation rules.
DATAVALUE_IMPORT	Import of data values.
EVENT_IMPORT	Import of events.
METADATA_IMPORT	Import of metadata.
DATAINTEGRITY	Processing of data integrity checks.

You can poll tasks through a GET request to the system tasks resource:

```
/api/25/system/tasks/{task-category-id}
```

A request to poll for the status of a data value import task looks like this:

```
/api/25/system/tasks/DATAVALUE_IMPORT
```

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
    }
]
```

1.42.5. Get appearance information

You can retrieve the available flag icons in JSON format with a GET request:

```
/api/25/system/flags
```

You can retrieve the available UI styles in JSON format with a GET request:

```
/api/25/system/styles
```

1.43. Locales

DHIS2 supports translations both for the user interface and for database content.

1.43.1. 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/25/locales/ui
```

1.43.2. Database content locales

You can retrieve and create locales for database content with GET and POST requests through the following resource. XML and JSON resource representations are supported.

```
/api/25/locales/db
```

1.44. Translations

DHIS2 allows for translations of database content. You can work with translations through the Web API using the translations resource.

```
/api/25/translations
```

1.44.1. Create 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/25/translations
```

The properties which support translations are listed in the table below.

Table 1.81. Property names

Property name	Description
name	Object name
shortName	Object short name

Property name	Description
description	Object description

The classes which support translations are listed in the table below.

Table 1.82. 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
TrackedEntityAttributeGroup	Tracked entity attribute group
TrackedEntity	Tracked entity
RelationshipType	Relationship type for tracked entity instances
OptionSet	Option set
Attribute	Attribute for metadata

1.44.2. Get translations

You can browse all translations through the translations resource:

```
GET /api/25/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/25/translations.json?fields=*&filter=className:eq:DataElement&filter=locale:eq:es
```

To get translations for a specific object for all properties:

```
/api/25/translations.json?
fields=*&filter=className:eq:DataElement&filter=locale:eq:fr&filter=objectId:eq:fbfJHSPpUQD
```

1.45. Short Message Service (SMS)

This section covers the SMS Web API for sending and receiving short text messages.

1.45.1. Outbound SMS service

The Web API supports sending outgoing SMS using the POST method. SMS can be sent to a 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/25/sms/outbound
```

```
{

  "message":"Sms Text",

  "recipients": [
    "47XXXXXX1",
    "47XXXXXX2"
  ]

}
```

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/25/sms/outbound?message=text&recipient=47XXXXXX
```

1.45.1.1. Gateway response codes

Gateway may response with following response codes.

Table 1.83. 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

Response code	Response Message	Detail Description
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
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

1.45.2. 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/25/sms/inbound
```

```
{
  "text": "sample text",
  "originator": "47XXXXXXXX",
  "gatewayid": "unknown",
  "receiveddate": "2016-05-01",
  "sentdate":"2016-05-01",
  "smsencoding": "1",
  "smsstatus":"1"
}
```

The Web API also supports a query parameter-based version.

```
/api/25/sms/inbound?message=text&originator=47XXXXXX&gateway=clickatel
```

Table 1.84. 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.

1.45.3. 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/25/sms/gateways
```

Configurations can also be retrieved for a specific gateway type using GET method.

```
GET /api/25/sms/gateways/{uid}
```

Configurations can be removed for specific gateway type using DELETE method.

```
DELETE /api/25/sms/gateways/{uid}
```

Default gateway can be retrieved with the GET method.

```
GET /api/25/sms/gateways/default
```

Default gateway can be set using the PUT method.

```
PUT /api/25/sms/gateways/default/{uid}
```

1.45.4. 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.

Clickatell

```
{
  "name" : "clickatell",
  "username": "clickatelluser",
  "password": "abc123",
  "Auth-token": "XXXXXXXXXXXXXXXXXXXX",
}
```

Bulksms

```
{
  "name": "bulkSMS",
  "username": "bulkuser",
  "password": "abc123",
}
```

GenericHttp

```
{
  "name" : "generic",
  "messageParameter": "message",
  "recipientParameter": "msisdn",
  "urlTemplate": "http://localhost:template",
  "parameters": [
    {
      "key": "username",
      "value": "user12"
      "classified": "false"
    },
    {
      "key": "password",
      "value": "XXX"
      "classified": "true"
    }
  ]
}
```

HTTP.OK will be returned if configurations are saved successfully. In all other cases HTTP.ERROR will be retured.

The various gateway configurations can be instantiated using the endpoints listed below.

Table 1.85. Gateway api end points

Gatway Type	Method	API End Points
Clickatell	POST/PUT	/api/sms/gateways/clickatell
Bulksms	POST/PUT	/api/sms/gateways/bulksms
Generichttp	POST/PUT	/api/sms/gateways/generichttp

1.46. SMS Commands

SMSCommands are being used to collect data through SMS. These commands belong to specific Parser Type. Each parser has different functionality.

1.46.1. API End Points

The list of commands can be retrieved using GET.

```
GET /api/25/sms/commands
```

One particular command can also be retreived using GET. Where parameter type can be one of the below mentioned parsers

```
GET /api/25/sms/commands/{commandName}?type=EVENT_REGISTRATION_PARSER
```

- KEY_VALUE_PARSER
- J2ME_PARSER
- ALERT_PARSER
- UNREGISTERED_PARSER
- TRACKED_ENTITY_REGISTRATION_PARSER
- PROGRAM_STAGE_DATAENTRY_PARSER
- EVENT_REGISTRATION_PARSER

1.47. Program Messages

Program message lets you send messages to tracked entity instances, contact addresses assiociated with organisation units, phone numbers and email addresses. You can send messages through the messages resource.

```
/api/25/messages
```

1.47.1. 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/25/messages
```

```json
{
    "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.

Table 1.86. 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.
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 DHIS 2.	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/25/messages"
  -H "Content-Type:application/json" -u admin:district -v
```

```
{
    "programMessages": [{
        "recipients": {
            "trackedEntityInstance": {
                "id": "PQfMcpmXeFE"
            }
        },
        "programInstance": {
            "id": "JMgRZyeLWOo"
        },
        "deliveryChannels": [
            "SMS"
        ],
        "text": "Please make a visit on Thursday"
    }]
}
```

1.47.2. Retrieving and deleting program messages

The list of messages can be retrieved using GET.

```
GET /api/25/messages
```

One particular message can also be retreived using GET.

```
GET /api/25/messages/{uid}
```

Message can be deleted using DELETE.

```
DELETE /api/25/messages/{uid}
```

1.47.3. 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 retreiving information.

Table 1.87. Query program messages API

Parameter	URL
programInstance	/api/25/messages?programInstance=6yWDMa0LP7
programStageInstance	/api/25/messages?programStageInstance=SllsjpfLH2
trackedEntityInstance	/api/25/messages?trackedEntityInstance=xdfejpfLH2
organisationUnit	/api/25/messages?ou=Sllsjdhoe3
processedDate	/api/25/messages?processedDate=2016-02-01

1.48. Users

This section covers the user resource methods.

```
/api/25/users
```

1.48.1. 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.

Table 1.88. 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.
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/25/users?query=konan&authSubset=true&pageSize=10
```

1.48.2. User credentials query

An alternative to the previous user query, is to directly query the user credentials (the part where username, etc resides) using /api/userCredentials endpoint, it supports all regular field and object filters as the other endpoints.

Get user credentials where username is admin:

```
/api/25/userCredentials?filter=username:eq:admin
```

Get username and code from all user credentials where username starts with adm:

```
/api/25/userCredentials?fields=username,code&filter=username:^like:adm
```

1.48.3. User account create and update

Both creating and updating a user is supported through the web-api. The payload itself is similar to other payloads in the web-api, so they support collection references etc. A simple example payload to create would be, the password should be sent in plain text (remember to only use this on a SSL enabled server) and will be encrypted on the backend:

```
{
  "firstName": "John",
  "surname": "Doe",
  "email": "johndoe@mail.com",
  "userCredentials": {
    "username": "johndoe",
    "password": "your-password-123",
    "userRoles": [ {
      "id": "Euq3XfEIEbx"
    } ]
  },
  "organisationUnits": [ {
    "id": "ImspTQPwCqd"
  } ],
  "userGroups": [ {
    "id": "vAvEltyXGbD"
  } ]
}
```

```
curl -X POST -u user:pass -d @u.json -H "Content-Type: application/json" http://
server/api/25/users
```

After the user is created, a Location header is sent back with the newly generated ID (you can also provide your own using /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 -u user:pass -d @u.json -H "Content-Type: application/json" http://server/
api/25/users/ID
```

For more info about the full payload available, please see /api/schemas/user

1.48.4. 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/25/users/invite" -H "Content-Type:application/
json" -u admin:district -v
```

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/25/users/invites" -H "Content-Type:application/
json"
  -u admin:district
```

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.
- The user to be invited must not be granted user roles with critical authorities (see below).
- 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.

The critical authorities which cannot be granted with invites include:
- ALL
- Scheduling administration
- Set system settings
- Add, update, delete and list user roles
- Add, update, delete and view SQL views

1.48.5. 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": "replica",
  "password": "Replica.1234"
}
```

This payload can be posted to the replica resource, where you provide the identifier of the user to replicate in the URL:

```
/api/25/users/<uid>/replica
```

An example of replicating a user using curl looks like this:

```
curl -d @replica.json "localhost/api/25/users/N3PZBUlN8vq/replica"
  -H "Content-Type:application/json" -u admin:district -v
```

1.49. Current user information and associations

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
```

Lists all messages and interpretations in the inbox (including replies):

```
/api/me/inbox
```

Gives the full profile information for current user. This endpoint support both GET to retrieve profile and POST to update profile (the exact same format is used):

```
/api/me/user-account
```

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>
```

Lists all organisation units directly assigned to the user:

```
/api/me/organisationUnits
```

Gives all the datasets assigned to the users organisation units, and their direct children. This endpoint contains all required information to build a form based on one of our datasets. If you want all descendants of your assigned organisation units, you can use the query parameter includeDescendants=true :

```
/api/me/dataSets
```

Gives all the programs assigned to the users organisation units, and their direct children. This endpoint contains all required information to build a form based on one of our datasets. If you want all descendants of your assigned organisation units, you can use the query parameter includeDescendants=true :

```
/api/me/programs
```

Gives the data approval levels which are relenvant to the current user:

```
/api/me/dataApprovalLevels
```

1.50. 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/25/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/25/systemSettings/my-key" -d "My long value"
```

```
 -H "Content-Type: text/plain" -u admin:district -v
```

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"
}
```

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/25/systemSettings/my-key
```

Alternatively, you can specify the key as a query parameter:

```
/api/25/systemSettings?key=my-key
```

You can retrieve specific system settings as JSON by repeating the key query parameter:

```
curl "play.dhis2.org/demo/api/25/systemSettings?
key=keyApplicationNotification&key=keyApplicationIntro"
  -H "Content-Type: application/json" -u admin:district -v
```

You can retrieve all system settings with a GET request:

```
/api/25/systemSettings
```

To delete a system 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.

Table 1.89. System settings

Key	Description
keyMessageEmailNotification	Send email notification for messages
keyMessageSmsNotification	Send sms notification for messages
keyUiLocale	Locale for the user interface
keyDbLocale	Locale for the database
keyAnalysisDisplayProperty	The property to display in analysis. Default: "name"
keyCurrentDomainType	Not yet in use
keyAutoSaveCaseEntryForm	Autosave case entry forms
keyAutoSaveDataEntryForm	Autosave data entry forms
keyTrackerDashboardLayout	Used by tracker capture
keyAutoSavetTrackedEntityForm	Autosave tracked entity forms
applicationTitle	The application title. Default: "DHIS2"
keyApplicationIntro	The application introduction
keyApplicationNotification	Application notification
keyApplicationFooter	Application left footer
keyApplicationRightFooter	Application right footer
keyFlag	Application flag
keyFlagImage	Flag used in dashboard menu

Key	Description
startModule	The startpage of the application. Default: "dhis-web-dashboard-integration"
factorDeviation	Data analysis standard deviation factor. Default: "2d"
keyEmailHostName	Email server hostname
keyEmailPort	Email server port
keyEmailTls	Use TLS. Default: "true"
keyEmailSender	Email sender
keyEmailUsername	Email server username
keyEmailPassword	Email server password
keyInstanceBaseUrl	The base url of the application instance
keySmsConfig	SMS configuration
keyCacheStrategy	Cache strategy. Default: "CACHE_6AM_TOMORROW"
phoneNumberAreaCode	Phonenumber area code
multiOrganisationUnitForms	Enable multi-organisation unit forms. Default: "false"
keyAccountRecovery	Enable user account recovery. Default: "false"
googleAnalyticsUA	Google Analytic UA key for tracking site-usage
credentialsExpires	Require user account password change. Default: "0" (Never)
keySelfRegistrationNoRecaptcha	Do not require recaptcha for self registration. Default: "false"
keyOpenIdProvider	OpenID provider
keyCanGrantOwnUserAuthorityGroups	Allow users to grant own user roles. Default: "false"
keyHideUnapprovedDataInAnalytics	Hide unapproved data in analytics. Default: "false"
keyAnalyticsMaxLimit	Maximum number of analytics recors. Default: "50000"
keyAnalyticsMaintenanceMode	Put analytics in maintenance mode. Default: "false"
keyDatabaseServerCpus	Number of database server CPUs. Default: "0" (Automatic)
helpPageLink	Link to help page. Default: "https://dhis2.github.io/dhis2-docs/master/en/user/html/dhis2_user_manual_en.html
keyAcceptanceRequiredForApproval	Acceptance required before approval. Default: "false"
keySystemNotificationsEmail	Where to email system notifications
keyAnalysisRelativePeriod	Default relative period for analysis. Default: "LAST_12_MONTHS"

Key	Description
keyRequireAddToView	Require authority to add to view object lists. Default: "false"
keyAllowObjectAssignment	Allow assigning object to related objects during add or update. Default: "false"
keyUseCustomLogoFront	Enables the usage of a custom logo on the frontpage. Default: "false"
keyUseCustomLogoBanner	Enables the usage of a custom banner on the website. Default: "false"
keyDataImportStrictPeriods	Require periods to match period type of data set. Default: "false"
keyDataImportStrictCategoryOptionCombos	Require category option combos to match category combo of data element. Default: "false"
keyDataImportStrictOrganisationUnits	Require organisation units to match assignment of data set. Default: "false"
keyDataImportStrictAttributeOptionsCombos	Require attribute option combis to match category combo of data set. Default: "false"
keyDataImportRequireCategoryOptionCombo	Require category option combo to be specified. Default: "false"
keyDataImportRequireAttributeOptionCombo	Require attribute option combo to be specified. Default: "false"
keyCustomJs	Custom JavaScript to be used on the website
keyCustomCss	Custom CSS to be used on the website
keyCalendar	The calendar type. Default: "iso8601".
keyDateFormat	The format in which dates should be displayed. Default: "yyyy-MM-dd".
appStoreUrl	The url used to point to the app store. Default: "https://www.dhis2.org/appstore"
keyStyle	The style used on the DHIS2 webpages. Default: "light_blue/light_blue.css".
keyRemoteInstanceUrl	Url used to connect to remote instance
keyRemoteInstanceUsername	Username used to connect to remote DHIS2 instance
keyRemoteInstancePassword	Password used to connect to remote DHIS2 instance
keyMapzenSearchApiKey	Key for the Mapzen geo search API

1.51. 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/25/userSettings
```

User settings not set by the user, will fall back to the equvalent system setting. To only return the values set explicitly by the user, you can append ?useFallback=false to the above URL, like this:

```
/api/25/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/25/userSettings/my-key?value=my-val
```

You can specify the user for which to save the setting explicitly with this syntax:

```
/api/25/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/25/userSettings/my-key" -d "My long value"
  -H "Content-Type: text/plain" -u admin:district -v
```

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/25/userSettings/keyUiLocale?value=fr" -X POST -u
 admin:district -v
```

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/25/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.

Table 1.90. 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 dasboard layout.

1.52. Organisation units

The organisationUnits resource follows the standard conventions as other metadata resources in DHIS2. This resource supports some additional query parameters.

1.52.1. Get list of organisation units

To get a list of organisation units you can use the following resource.

`/api/25/organisationUnits`

Table 1.91. 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.

1.52.2. Get organisation unit with relations

To get an organisation unit with related organisation units you can use the following resource.

`/api/25/organisationUnits/{id}`

Table 1.92. 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 subhierarchy.
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 subhierarchy (relative to the organisation unit where the immediate level below is level 1).

1.53. Data sets

The dataSets resource follows the standard conventions as other metadata resources in DHIS2. This resource supports some additional query parameters.

`/api/25/dataSets`

To retrieve the version of a data set you can issue a GET request:

```
GET /api/25/dataSets/<uid>/version
```

To bump (increase by one) the version of a data set you can issue a POST request:

```
POST /api/25/dataSets/<uid>/version
```

1.54. 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/25/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/25/filledOrganisationUnitLevels" -H "Content-
Type:application/json"
  -d @levels.json -u admin:district -v
```

1.55. Static content

The staticContent resource allowes 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/25/staticContent
```

Table 1.93. 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/25/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/25/
staticContent/logo_front"
  -X POST -H "Content-Type: multipart/form-data" -u admin:district -v
```

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/25/staticContent/<key>
```

Example request to retrieve the file stored for logo_front:

```
curl "https://play.dhis2.org/demo/api/25/staticContent/logo_front" -L -X GET -u
 admin:district -v
```

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.

1.56. 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/25/configuration
```

You can send GET and POST requests to the following specific resources:

```
GET /api/25/configuration/systemId
```

```
GET POST DELETE /api/25/configuration/feedbackRecipients
```

```
GET POST DELETE /api/25/configuration/offlineOrganisationUnitLevel
```

```
GET POST /api/25/configuration/infrastructuralDataElements
```

```
GET POST /api/25/configuration/infrastructuralIndicators
```

```
GET POST /api/25/configuration/infrastructuralPeriodType
```

```
GET POST DELETE /api/25/configuration/selfRegistrationRole
```

```
GET POST DELETE /api/25/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/25/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.

Table 1.94. Configuration values

Configuration property	Value
feedbackRecipients	User group ID

Configuration property	Value
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/25/configuration/feedbackRecipients" -d "wl5cDMuUhmF"
  -H "Content-Type:text/plain"-u admin:district -v
```

1.57. Internationalization

In order to retrieve key-value pairs for translated strings you can use the i18n resource.

```
/api/25/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/25/i18n" -X POST
  -H "Content-Type: application/json" -u admin:district -v
```

The result will look like this:

```
{
  "access_denied":"Access denied",
  "uploading_data_notification":"Uploading locally stored data to the server"
}
```

1.58. 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 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
```

Table 1.95. Query parameters

Query parameter	Required	Description
svg	Yes	The SVG content
filename	No	The file name for the returned attachment without file extension

1.59. Tracked entity instance management

Tracked entity instances have full CRUD (create, read, update, delete) support in the Web-API. Together with the API for enrollment most operations needed for working with tracked entity instances and programs are supported.

```
/api/25/trackedEntityInstances
```

1.59.1. 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",
    "attributes": [ {
        "attribute": "attribute-id",
        "value": "attribute-value"
    } ]
}
```

For getting the IDs for relationship, 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 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 -v
```

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"
        }
      ]
    }
  ]
}
```

1.59.2. Updating a tracked entity instance

For updating a tracked entity instance, the payload is the 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>
```

1.59.3. Deleting a tracked entity instance

To delete a tracked entity instance you can make a request to the URL identifiying the tracked entity instance with the HTTP DELETE method. The URL is equal to the one above used for update.

1.59.4. 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/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/enrollments/<enrollment-id>/cancelled
```

For completing a enrollment for a tracked entity instance you can make a PUT request to the following URL:

```
/api/enrollments/<enrollment-id>/completed
```

For deleting a enrollment, you can make a DELETE request to the following URL:

```
/api/enrollments/<enrollment-id>
```

1.59.5. Update strategies

Two update strategies for tracked entity instance 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.

Table 1.96. 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/trackedEntityInstances?strategy=CREATE_AND_UPDATE
```

1.59.6. 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 instane.

```
curl -X POST -d @tei.json -H "Content-Type: application/json"
  -u user:pass http://server/api/25/trackedEntityInstances
```

1.59.7. Generated tracked entity instance attributes

Tracked entity instance attributes that is using automatic generation of unique values has two endpoints that is used by apps. One is used to generate values, the other to generate and reserve one or more values for use later.

1.59.7.1. Generate value endpoint

Online web apps and other clients that wants 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 by use of the generateAndReseve endpoint.

```
curl -X GET -H "Content-Type: application/json" -u admin:district
        https://play.dhis2.org/dev/api/trackedEntityAttributes/lZGmxYbs97q/generate
```

1.59.7.2. Generate and reserve value endpoint

The generate and reserve endpoint is used by offline clients that needs 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 retreieve a number of tracked entity instance reserved values. An optional parameter numberToReserve specifies how many ids to generate (default is 1).

```
curl -X GET -H "Content-Type: application/json" -u admin:district
        https://play.dhis2.org/dev/api/trackedEntityAttributes/lZGmxYbs97q/
generateAndReserve?numberToReserve=3
```

```
[
    {
        "created":"2016-06-25T07:52:26.466+0000",
        "expiryDate":"2017-06-25T07:52:26.466+0000",
        "value":"43327401",
        "trackedEntityAttribute":{  }
    },
    {
        "created":"2016-06-25T07:52:26.468+0000",
        "expiryDate":"2017-06-25T07:52:26.468+0000",
        "value":"82676084",
        "trackedEntityAttribute":{  }
    },
    {
        "created":"2016-06-25T07:52:26.471+0000",
```

```
        "expiryDate":"2017-06-25T07:52:26.471+0000",
        "value":"91700776",
        "trackedEntityAttribute":{  }
    }
]
```

The generate and reserve endpoint returns an array of reserved values:

Table 1.97. Tracked entity reserved values

Property	Description
created	The date the reserved value was created.
expiryDate	After this date the value can be deleted/unreserved by the server if no tracked entity has been registered with the value. The deletion does not happen automatically at the exact time of the expiration, but can/will be done with scripts or mechanisms implemneted later.
value	The actual generated value that is reserved.
trackedEntityAttribute	The tracked entity attribute that the value was generated for.

1.60. Tracked entity instance query

To query for tracked entity instances you can interact with the /api/trackedEntityInstances resource.

```
/api/25/trackedEntityInstances
```

1.60.1. Request syntax

Table 1.98. 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 idenfiers, separated by ";".
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 identifer. Restricts instances to the given tracked instance type.
page	The page number. Default page is 1.

Query parameter	Description
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.

The available organisation unit selection modes are explained in the following table.

Table 1.99. 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 and all children, i.e. all organisation units in the subhierarchy.
ACCESSIBLE	The data view organisation units associated with the current user and all children, i.e. all organisation units in the subhierarchy. 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 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/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 descendants organisation unit query mode:

```
api/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 us used as a filter:

```
api/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/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/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/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
&ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
api/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/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/trackedEntityInstances.json?filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu
&ouMode=DESCENDANTS&page=2&pageSize=3
```

You can use a range of operators for the filtering:

Table 1.100. 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 ";"

1.60.2. 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=*):

```
{
    "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": "Mother maiden name",
                        "displayName": "Mother maiden name",
                        "attribute": "o9odfev2Ty5",
                        "type": "string",
                        "value": "Gabriel"
                    },
                    {
                        "code": "National identifier",
                        "displayName": "National identifier",
                        "attribute": "AuPLng5hLbE",
                        "type": "string",
                        "value": "465700042"
                    },
                    {
                        "code": "Occupation",
                        "displayName": "Occupation",
                        "attribute": "A4xFHyieXys",
                        "type": "string",
                        "value": "Biophysicist"
                    },
                    {
                        "code": "Company",
                        "displayName": "Company",
                        "attribute": "kyIzQsj96BD",
                        "type": "string",
                        "value": "Sav-A-Center"
                    },
                    {
                        "code": "Vehicle",
                        "displayName": "Vehicle",
                        "attribute": "VHfUeXpawmE",
                        "type": "string",
                        "value": "2008 Citroen Picasso"
                    },
                    {
                        "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"
                }
            ]
        }
    ]
}
```

1.61. 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/24/trackedEntityInstances/query
```

1.61.1. Request syntax

Table 1.101. 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 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 idenfiers, separated by ";".
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.

Query parameter	Description
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 identifer. 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.
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.

The available organisation unit selection modes are explained in the following table.

Table 1.102. Organisation unit selection modes

Mode	Description
SELECTED	Organisation units defined in the request.
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.

Note that you can specify attributes with filters for constraining the instances to return, or attributes without filters in order to include the attribute in the response without any constraints. Attributes will be included in the response, while filters will only be used as criteria.

Certain rules apply to which attributes are defined when no attributes are specified in the request:

• If not specifying a program, the attributes defined to be displayed in lists with no program will be included in the response.

• If specifying a program, the attributes linked to the program 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/25/trackedEntityInstances/query.json?ou=DiszpKrYNg8
```

A query on all attributes for a specific value and organisation unit, using an exact word match:

```
/api/25/trackedEntityInstances/query.json?query=scott&ou=DiszpKrYNg8
```

A query on all attributes for a specific value, using a partial word match:

```
/api/25/trackedEntityInstances/query.json?query=LIKE:scott&ou=DiszpKrYNg8
```

You can query on multiple words separated by the the URL character for space which is %20, will use a logical AND query for each word:

```
/api/25/trackedEntityInstances/query.json?query=isabel%20may&ou=DiszpKrYNg8
```

A query where the attributes to include in the response are specified:

```
/api/25/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/25/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 us used as a filter:

```
/api/25/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/25/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/25/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/25/trackedEntityInstances/query.json?filter=zHXD5Ve1Efw:EQ:A
  &ou=O6uvpzGd5pu&ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
/api/25/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/25/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/25/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/25/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:

Table 1.103. 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 ";"

1.61.2. 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"]
    ]
}
```

1.62. Enrollment instance query

To query for tracked entity instances you can interact with the /api/enrollments resource.

```
/api/25/enrollments
```

1.62.1. Request syntax

Table 1.104. Tracked entity instances query parameters

Query parameter	Description
ou	Organisation unit idenfiers, 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 identifer. 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.

The available organisation unit selection modes are explained in the following table.

Table 1.105. 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.

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.

A query for all instances associated with a specific organisation unit can look like this:

```
/api/25/enrollments.json?ou=DiszpKrYNg8
```

To constrain the response to instances which are part of a specific program you can include a program query parameter:

```
/api/25/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&program=ur1Edk5Oe2n
```

To specify program enrollment dates as part of the query:

```
/api/25/enrollments.json?
&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/25/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&trackedEntity=cyl5vuJ5ETQ
```

To constrain the response to instances of a specific tracked entity instance you can include a tracked entity instance query parameter, in this case we are restricted it to available enrollments viewable for current user:

```
/api/25/enrollments.json?ouMode=ACCESSIBLE&trackedEntityInstance=tphfdyIiVL6
```

By default the instances are returned in pages of size 50, to change this you can use the page and pageSize query parameters:

```
/api/25/enrollments.json?ou=O6uvpzGd5pu&ouMode=DESCENDANTS&page=2&pageSize=3
```

1.62.2. 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=*):

```
{
    "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"
        }
    ]
}
```

1.63. 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/25/trackedEntityInstasnces?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/25/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/25/events?strategy=DELETE"
```

1.64. 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 DHIS 2.

```
/api/25/email
```

1.64.1. 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/25/email/notification" -X POST -H "Content-
Type:application/json" -u admin:district -v
```

1.64.2. 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/25/email/test" -X POST -H "Content-Type:application/json" -u
 admin:district -v
```

1.65. 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/25/sharing
```

1.65.1. Get sharing status

To request the sharing status for an object use a GET request to:

```
/api/25/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-------"
            }
        ]
    }
}
```

1.65.2. 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/25/sharing?type=dataElement&id=fbfJHSPpUQD"
  -H "Content-Type:application/json" -u admin:district -v
```

1.66. Scheduling

To schedule tasks to run at fixed intervals you can interact with the scheduling resource. To configure tasks you can do a POST request to the following resource:

```
/api/25/scheduling
```

The payload in JSON format is described below.

```
{
    "resourceTableStrategy": "allDaily",
    "analyticsStrategy": "allDaily",
    "monitoringStrategy": "allDaily",
    "dataSynchStrategy": "enabled"
}
```

An example using curl:

```
curl "localhost/dhis/api/25/scheduling" -d @scheduling.json -X POST
  -H "Content-Type:application/json" -u admin:district -v
```

The table below lists available task strategies.

Table 1.106. Task strategies

Task	Strategies
Resource table task	allDaily | allEvery15Min
Analytics task	allDaily | last3YearsDaily
Monitoring	allDaily
Data synch task	enabled

1.67. 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
```

1.68. UI customization

To customize the UI of the DHIS2 application you can insert custom Javascript and CSS styles through the files resource. 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.

1.68.1. 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/25/files/script"
  -H "Content-Type:application/javascript" -u admin:district -v
```

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/25/files/script
```

To remove the Javascript content you can use a DELETE-request.

1.68.2. 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/25/files/style"
  -H "Content-Type:text/css" -u admin:district -v
```

You can fetch the CSS content with a GET-request:

```
/api/25/files/style
```

To remove the Javascript content you can use a DELETE-e "request.

1.69. Synchronization

This section covers pull and push of data and metadata.

1.69.1. Data push

To initiate a data 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/25/synchronization/dataPush
```

1.69.2. 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/25/synchronization/metadataPull
```

1.69.3. 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/25/synchronization/availability
```

1.70. Apps

The /api/apps endpoint can be used for installing, deleting and listing apps. The app key is derived from the name of the ZIP archive, exluding the file extension.

```
/api/25/apps
```

1.70.1. Get apps

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 -X GET -u user:pass -H "Accept: application/json" http://server.com/api/25/apps
```

You can also simply point your web browser to the resource URL:

```
http://server.com/api/25/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/25/apps?
filter=appType:eq:DASHBOARD_APP&filter=name:ilike:youtube
```

App names support the eq and ilike filter operators, while appType supports eq only.

1.70.2. 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/25/apps
```

1.70.3. Delete an app

To delete an app, you can issue the following command:

```
curl -X DELETE -u user:pass http://server.com/api/25/apps/<app-key>
```

1.70.4. 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/25/apps
```

1.70.5. Configure apps

To let DHIS2 serve apps from the Web API make sure to set the "App base URL" to point to
the apps resource, i.e.:

```
http://server.com/api/25/apps
```

To set the apps configuration you can make a POST request to the config resourec with a JSON
payload:

```
{
  "appFolderPath": "/home/dhis/config/apps",
  "appBaseUrl": "http://server.com/api/apps"
}
```

```
curl -X POST -u user:pass -d @config.json http://server.com/api/25/apps/config
```

To restore the default app settings you can make a DELETE request to the config resource:

```
curl -X DELETE -u user:pass http://server.com/api/25/apps/config
```

Note that by default apps will be served through the apps Web API resource, and the file system
folder will be DHIS2_HOME/apps. These systems should be fine for most situations.

1.71. 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/25/appStore
```

1.71.1. Get apps

You can retrieve apps with a GET request:

```
GET /api/25/appStore
```

A sample JSON response is described below.

```
{
  "name": "DHIS2 App Store",
  "description": "Send your apps with a description to us at apps@dhis2.org",
  "apps": [
    {
      "name": "Tabular Tracker Capture",
      "description": "Tabular Tracker Capture is an app that makes you more
 effective.",
      "developer": "John Develop",
      "img_large": "https://www.dhis2.org/images/appstore/tabular-tracker-
capture.png",
      "versions": [
        {
          "id": "upAPqrVgwK6",
          "version": "1.2",
          "min_platform_version": "2.17",
          "max_platform_version": "2.20",
          "download_url": "https://www.dhis2.org/download/appstore/tabular-tracker-
capture-12.zip",
```

```
            "demo_url": "http://play.dhis2.org/demo"
            }
        ]
      }
    ]
}
```

1.71.2. 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/25/appStore/{app-version-id}
```

1.72. Data store

Using the dataStore resource, developers can store arbitrary data for their apps. 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/25/dataStore
```

1.72.1. 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.

Table 1.107. Data store structure

Item	Description	Data type
Namespace	Namespace for organization of entries.	String
Key	Key for identifiaction 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

1.72.2. Get keys and namespaces

For a list of all existing namespaces:

```
GET /api/25/dataStore
```

Example curl request for listing:

```
curl "play.dhis2.org/demo/api/25/dataStore" -X GET -u admin:district -v
```

Example response:

```
[
    "foo",
    "bar"
```

```
]
```

For a list of all keys in a namespace:

```
GET /api/25/dataStore/<namespace>
```

Example curl request for listing:

```
curl "play.dhis2.org/demo/api/25/dataStore/foo" -X GET -u admin:district -v
```

Example response:

```
[
    "key_1",
    "key_2"
]
```

To retrieve a value for an existing key from a namespace:

```
GET /api/25/dataStore/<namespace>/<key>
```

Example curl request for retrieval:

```
curl "play.dhis2.org/demo/api/25/dataStore/foo/key_1" -X GET -u admin:district -v
```

Example response:

```
{
    "foo":"bar"
}
```

To retrieve meta-data for en existing key from a namespace:

```
GET /api/25/dataStore/
```

Example curl request for retrieval:

```
curl "play.dhis2.org/demo/api/25/dataStore/foo/key_1/metaData" -X GET -u
 admin:district -v
```

Example response:

```
{
    "created": "...",
    "user": {...},
    "namespace": "foo",
    "key": "key_1"
}
```

1.72.3. Create values

To create a new key and value for a namespace:

```
POST /api/25/dataStore/<namespace>/<key>
```

Example curl request for create, assuming a valid json payload:

```
curl "https://play.dhis2.org/demo/api/25/dataStore/foo/key_1" -X POST
  -H "Content-Type: application/json" -d "{\"foo\":\"bar\"}" -u admin:district -v
```

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/25/dataStore/<namespace>/<key>?encrypt=true
```

1.72.4. Update values

To update a key that exists in a namespace:

```
PUT /api/25/dataStore/<namespace>/<key>
```

Example curl request for update, assuming valid JSON payload:

```
curl "https://play.dhis2.org/demo/api/25/dataStore/foo/key_1" -X PUT -d "[1, 2, 3]"
  -H "Content-Type: application/json" -u admin:district -v
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Key 'key_1' updated."
}
```

1.72.5. Delete keys

To delete an existing key from a namespace:

```
DELETE /api/25/dataStore/<namespace>/<key>
```

Example curl request for delete:

```
curl "play.dhis2.org/demo/api/25/dataStore/foo/key_1" -X DELETE -u admin:district -v
```

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/25/dataStore/<namespace>
```

Example curl request for delete:

```
curl "play.dhis2.org/demo/api/25/dataStore/foo" -X DELETE -u admin:district -v
```

Example response:

```
{
    "httpStatus": "OK",
    "httpStatusCode": 200,
    "status": "OK",
    "message": "Namespace 'foo' deleted."
}
```

1.73. 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/25/userDataStore
```

1.73.1. User data store structure

userDataStore consists of a user, a namespace, keys and associated values. The combination of user, namespace and key is unique.

Table 1.108. 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

1.73.2. Get namespaces

Returns an array of all existing namespaces

```
GET /api/25/userDataStore
```

Example request:

```
curl -X GET -H "Content-Type: application/json" -u admin:district "play.dhis2.org/
api/25/userDataStore" -v
```

```
[
  "foo",
  "bar"
]
```

1.73.3. Get keys

Returns an array of all existing keys in a given namespace

```
GET /api/userDataStore/<namespace>
```

Example request:

```
curl -X GET -H "Content-Type: application/json" -u admin:district "play.dhis2.org/
api/25/userDataStore/foo" -v
```

```
[
  "key_1",
  "key_2"
]
```

1.73.4. Get value

Returns the value for a given namespace and key

```
GET /api/25/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X GET -H "Content-Type: application/json" -u admin:district "play.dhis2.org/
api/25/userDataStore/foo/bar"
```

```
{
  "some": "value"
}
```

1.73.5. Create value

Adds a new value to a given key in a given namespace.

```
POST /api/25/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X POST -H "Content-Type: application/json" -u admin:district -d "['some value']"
  "play.dhis2.org/api/25/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 credetials and such) you can append a query to the url like this:

```
GET /api/25/userDataStore/<namespace>/<key>?encrypt=true
```

1.73.6. Update existing value

Updates an existing value

```
PUT /api/25/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X PUT -H "Content-Type: application/json" -u admin:district -d "['new value']"
  "play.dhis2.org/api/25/userDataStore/foo/bar"
```

```
{
  "httpStatus":"Created",
  "httpStatusCode":201,
  "status":"OK",
  "message":"Key 'bar' in namespace 'foo' updated."
}
```

1.73.7. Delete key

Delete a key

```
DELETE /api/25/userDataStore/<namespace>/<key>
```

Example request:

```
curl -X DELETE -u admin:district "play.dhis2.org/api/25/userDataStore/foo/bar"
```

```
{
  "httpStatus":"OK",
  "httpStatusCode":200,
  "status":"OK",
  "message":"Key 'bar' deleted from the namespace 'foo."
}
```

1.73.8. Delete namespace

Delete all keys in the given namespace

```
DELETE /api/25/userDataStore/<namespace>
```

Example request:

```
curl -X DELETE -u admin:district "play.dhis2.org/api/25/userDataStore/foo"
```

```
{
  "httpStatus":"OK",
  "httpStatusCode":200,
  "status":"OK",
  "message":"All keys from namespace 'foo' deleted."
}
```

1.74. Predictors

A predictor allows you to generate data values based on an expression. This can be used to generate targets, thresholds and estimated values. You can interact with predictors through the /api/25/predictors resource.

```
/api/25/predictors
```

1.74.1. Creating a predictor

You can create a predictor with a POST request to the predictors resource:

```
POST /api/25/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})",
        "dataElements": [],
        "sampleElements": [{
            "id": "r6nrJANOqMw"
        }]
    },
    "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 predited data values. The generator element refers to the expression to use when calculating the predicted values.

1.74.2. Generating predicted values

To run all predictors (generating predicted values) you can make a POST request to the run resource:

```
POST /api/25/predictors/run
```

To run a single predictor you can make a POST request to the run resource for a predictor:

```
POST /api/25/predictors/AG10KUJCrRk/run
```

To do a dry run for a single predictor you can use the dryRun resource. This resource will return all predicted values in the response, as opposed to actually writing those values to the database:

```
POST /api/25/predictors/AG10KUJCrRk/dryRun
```

1.75. Tokens

The tokens resource provides access tokens to various services.

1.75.1. 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.

Table 1.109. 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.

1.76. 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 -v
```

Chapter 2. Apps

A packaged app is an Open Web App that has all of its resources (HTML, CSS, JavaScript, app manifest, and so on) contained in a zip file. It can be uploaded to a DHIS2 installation directly through the user interface at runtime. A packaged app is a ZIP file with an app manifest in its root directory. The manifest must be named `manifest.webapp`. A throrough description of apps can be obtained here.

2.1. Purpose of packaged Apps

The purpose of packaged apps is to extend the web interface of DHIS2, without the need to modify the source code of DHIS2 itself. A system deployment will often have custom and unique requirements. The apps provide a convenient extension point to the user interface. Through apps, you can complement and customize the DHIS2 core functionality with custom solutions in a loosely coupled and clean manner.

Apps do not have permissions to interact directly with DHIS2 Java API. Instead, apps are expected to use functionality and interact with the DHIS2 services and data by utilizing the DHIS2 Web API.

2.2. Creating Apps

DHIS2 apps are constructed with HTML, JavaScript and CSS files, similar to any other web application. Apps also need a special file called manifest.webapp which describes the contents of the app. This file should be in the format specified by the W3C Manifest for Web Applications. A basic example of the manifest.webapp is shown below:

```
{
    "version": "0.1",
    "name": "My App",
    "description": "My App is a Packaged App",
    "launch_path": "/index.html",
    "appType": "APP",
    "icons": {
        "16": "/img/icons/mortar-16.png",
        "48": "/img/icons/mortar-48.png",
        "128": "/img/icons/mortar-128.png"
    },
    "developer": {
        "name": "Me",
        "url": "http://me.com"
    },
    "default_locale": "en",
    "activities": {
        "dhis": {
            "href": "*",
            "namespace": "my-namespace"
        }
    }
}
```

The manifest.webapp file must be located at the root of the project. Among the properties are:

- The icons→48 property is used for the icon that is displayed on the list of apps that are installed on a DHIS2 instance.
- The activities property is an dhis-specific extension meant to differentiate between a standard Open Web App and an app that can be installed in DHIS2.

- The * value for href is converted to the appropriate URL when the app is uploaded and installed in DHIS2. This value can then be used by the application's JavaScript and HTML files to make calls to the DHIS2 Web API and identify the correct location of DHIS2 server on which the app has been installed. To clarify, the activities part will look similar to this after the app has been installed:

```
"activities": {
    "dhis": {
        "href": "http://apps.dhis2.org/demo",
        "namespace": "my-namespace"
    }
}
```

The namespace property can be added if your app is utilizing the dataStore or userDataStore api. When adding the namespace property, only users with access to your app are allowed to make changes to the namespace. A namespace can only be reserved in this way once. If another app tries to reserve a namespace already in use, the installation of the other app will fail.

If you have a collection of apps that want to share the same namespace, but also wish to reserve it, the users of the apps needs to have the authority to use the app that initially reserved the namespace.

> ☞   Note
>
>   Namespaces will not be created until atleast one key-value pair is present in the namespace. Specifying a namespace in the manifest only restricts the access and does not create any data in the namespace.

The appType property specifies how the app will be displayed by the DHIS2 instance. The possible values for appType and their effects are explained in the following table.

Table 2.1. App types

App type	Description
APP	Will be listed in the "apps" menu
DASHBOARD_WIDGET	Available from the search box on the dashboard, can be added as an item on any dashboard
TRACKER_DASHBOARD_WIDGET	Can be embedded in the tracker dashboard (this type is not yet supported)
RESOURCE	Resource apps are packages that can be shared by multiple other apps. These apps are not shown anywhere in the UI, except from in the app management app.

If no appType is specified in the manifest, the system will use "APP" by default.

To read the JSON structure into JavaScript, you can use a regular AJAX request and parse the JSON into an object. Most Javascript libraries provide some support, for instance with jQuery it can be done like this:

```
$.getJSON( "manifest.webapp", function( json ) {
    var apiBaseUrl = json.activities.dhis.href + "/api";
} );
```

The app can contain HTML, JavaScript, CSS, images and other files which may be required to support it . The file structure could look something like this:

```
/
```

```
/manifest.webapp    #manifest file (mandatory)
/css/               #css stylesheets (optional)
/img/               #images (optional)
/js/                #javascripts (optional)
```

☞ Note

It is only the `manifest.webapp` file which must be placed in the root. It is up the developer to organize CSS, images and JavaScript files inside the app as needed.

All the files in the project should be compressed into a standard zip archive. Note that the manifest.webapp file must be located on the root of the zip archive (do not include a parent directory in the archive). The zip archive can then be installed into DHIS2 as you will see in the next section.

2.3. Installing Apps into DHIS2

Apps   can   be   installed   by   uploading   zip   file   into   the   App Manager.   In,   Services   →   Apps,   click   on   the   App   Store   menu   item.
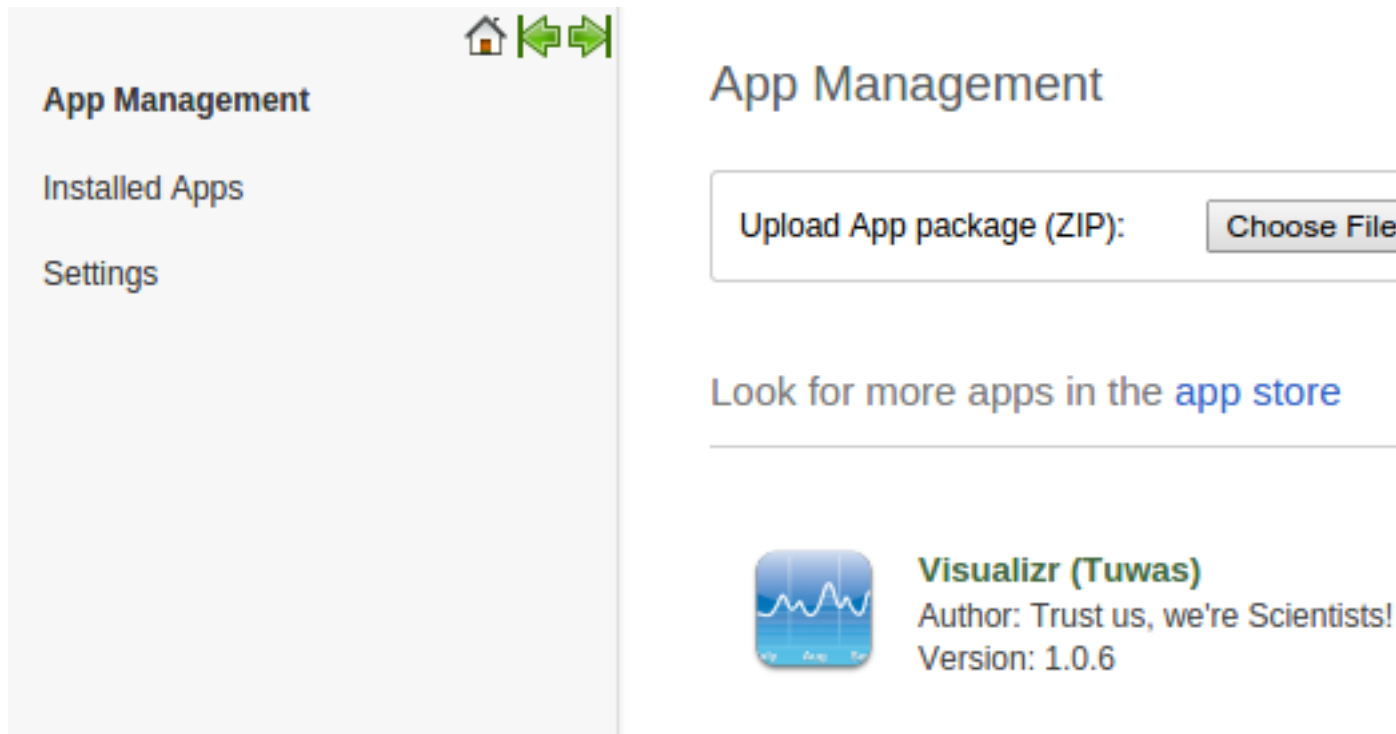
The app can be uploaded by pressing the Browse button and after selecting the zip package, the file is uploaded automatically and installed in DHIS2. You can also browse through apps in the DHIS2 app store and download apps from there. The DHIS2 app store allows for app searching, reviewing, commenting, requesting features, rating on the apps by the community.

2.4. Launching Apps

After installation, your apps will be integrated with the menu system and can be accessed under services and from the module overview page. It can also be accessed from the home page of the apps module. Click on an app in the list in order to launch it.

2.5. Adding the DHIS 2 menu to your app

In order to maintain a uniform appearance within DHIS2 it is possible to add your app's icon to the top menu. To begin, we start with a screenshot of the top-menu of the DHIS2 user interface, where your app's icon will be placed.
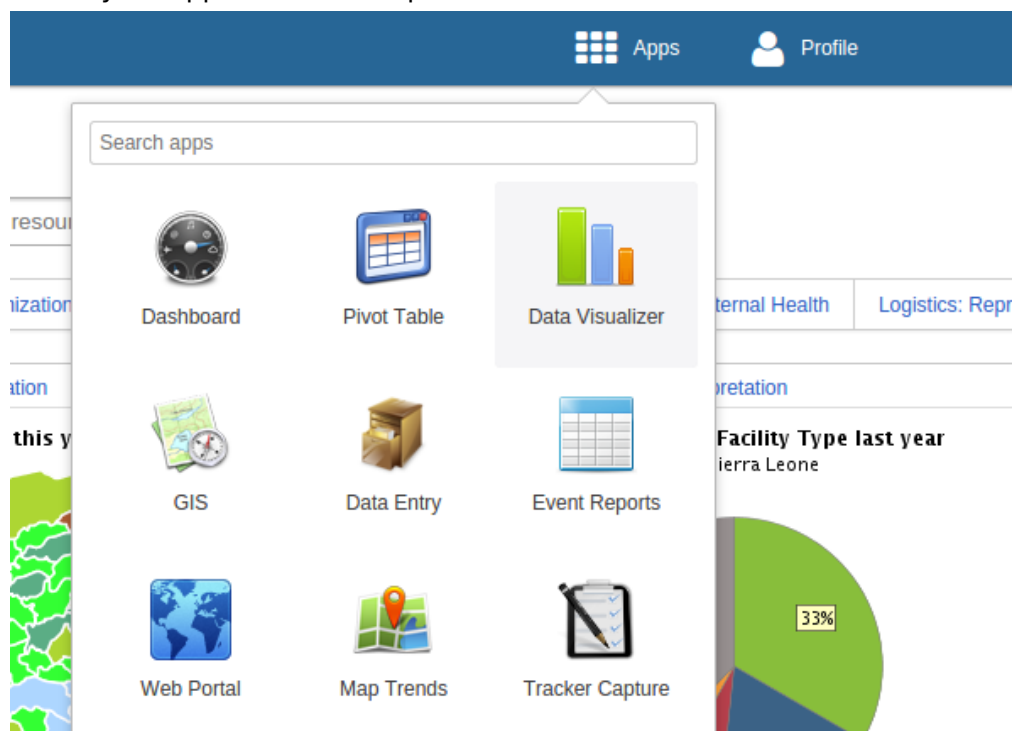
The first step is to adding the menu is including the style-sheets and scripts that are required. All JavaScript files are found in `/dhis-web-commons/javascripts/dhis2/` while the CSS files are found in `/dhis-web-commons/font-awesome/css/font-awesome.min.css` and `/dhis-web-commons/css/menu.css`

The following list provides a description of each file:

Scripts:

- `jquery.min.js / jqLite / angular.element` : One of the mentioned libraries needs to be present. DHIS2 employs a stripped-down version of `jqLite` that is present in Angular for the menu. This makes it compatible with jqLite and jQuery.

- `dhis2.translate.js` : Translate script that translates menu text to your current dhis language setting

- `dhis2.menu.js` : Menu logic that deals with all the ordering, searching of menu items etc.

- dhis2/dhis2.menu.ui.js : Menu ui code that has all the menu user interface related code for scrolling, shortcuts, HTML markup etc.

Stylesheets:

- `font-awesome.min.css` : Used for various icons in the menu.

- `menu.css` : The CSS used for the menu.

- `dhis2.translate.js` : Translate script that translates menu text to your current DHIS2 language setting

For a app that will run using the same URL structure as the normal DHIS2 apps, only the JavaScript files and style-sheets are required. If your app is running using a different URL structure than the default one, you will need to specify a *base URL* before including the menu scripts. Including the scripts and style-sheets would look something like the following:

```
<!-- DHIS2 Settings initialization for a baseUrl that is used for the menu -->
<script>
```

```
 window.dhis2 = window.dhis2 || {};
 dhis2.settings = dhis2.settings || {};
 dhis2.settings.baseUrl = 'dhis';
</script>


<!-- Menu scripts -->
 <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.translate.js">
</script>
 <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.menu.js">
</script>
 <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.menu.ui.js"></script>


<!-- Stylesheets related to the menu -->
<link type="text/css" rel="stylesheet" href="./dhis-web-commons/font-awesome/css/font-
awesome.min.css"/>
<link type="text/css" rel="stylesheet" media="screen" href="./dhis-web-commons/css/
menu.css">
```

To clarify, the following part will initialize some variables. If you do not use any other DHIS2 libraries these will not be set and therefore will have to be set by you first. After that the third line specifies a base URL of where your DHIS2 instance is running on your web server. For example: dhis in this case means the server is running at http://localhost:8080/dhis/. Note that you will only have to specify the part after the web address. So if your instance is running at http://www.example.com/myInstance/ you would only specify myInstance

```
<!-- Example setting for myInstance -->
<script>
window.dhis2 = window.dhis2 || {};
dhis2.settings = dhis2.settings || {};
dhis2.settings.baseUrl = 'myInstance';
</script>
```

The above will only include the necessary scripts to be able to show the menu. To actually make it show up we have two possibilities. The first one is using a basic <div> element with an id attribute.

```
<div id="dhisDropDownMenu"></div>
```

An alternative is available when your application uses angular. We have included a directive to show the menu. This would be used as follows:

```
<div d2-menu></div>
```

The element type in this case does not really matter. As long as you include the d2-menu directive. To be able to use the menu directive you would also have to include the directive in your angular app. The angular module containing the directive is called d2Menu.

```
'use strict';


var appMenu = angular.module('appMenu',
  ['ngRoute',
  'ngCookies',
  'd2Menu']);
```

The minimum amount of code to show the menu is shown below. You could use this as a starting reference.

```
<!DOCTYPE html>
<html ><!--ng-app="appMenu"> -->
    <head>
```

```
        <title>Dhis2 Menu</title>

        <!-- Stylesheets related to the menu -->
        <link type="text/css" rel="stylesheet" href="./dhis-web-commons/font-awesome/
css/font-awesome.min.css"/>
        <link type="text/css" rel="stylesheet" media="screen" href="./dhis-web-
commons/css/menu.css">
    </head>

    <body style="background-color: black;">

        <div id="dhisDropDownMenu"></div>

        <!-- DHIS2 Settings initialization for a baseUrl that is used for the menu -->
        <script>
            window.dhis2 = window.dhis2 || {};
            dhis2.settings = dhis2.settings || {};
            dhis2.settings.baseUrl = 'dhis';
        </script>

        <!-- Menu scripts -->
        <script type="text/javascript" src="./dhis-web-commons/javascripts/jQuery/
jquery.min.js"></script>
        <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.translate.js"></script>
        <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.menu.js"></script>
        <script type="text/javascript" src="./dhis-web-commons/javascripts/dhis2/
dhis2.menu.ui.js"></script>

    </body>
</html>
```

Chapter 3. Development Infrastructure

3.1. Release process

Checklist for release.

1. In order to tag the source source code with new release. First temporarily add a dependency to dhis-web in the root pom.xml:

```
<module>dhis-web</module>
```

Use the mvn version plugin with:

```
mvn versions:set
mvn versions:commit
```

This will prompt you to enter the version. Remove the dhis-web dependency. Update application cache manifests in the various apps to new version. Commit the changes to trunk.

2. Push a release branch to Launchad, e.g. with:

```
bzr push lp:~dhis2-devs-core/dhis2/2.19
```

3. Tag source code with SNAPSHOT release. Remember to add the temporary dhis-web dependency in the root pom.xml. Commit the changes to trunk.

4. Enable email notifications for release branch.

5. Create Jenkins build for the release WAR file.

6. Create Jenkins build for the release Javadocs.

7. Create automatic copy job for WAR file from Jenkins to dhis2.org.

8. Create automatic WAR file update of play.dhis2.org/demo and play.dhis2.org/dev systems.

9. Update the database and WAR file on play.dhis2.org/demo and play.dhis2.org/dev instances. To create a dump in directory format and turn it into a tar file:

```
pg_dump dhis2 -Fd -U dhis -j 4 -f db.pgdump
tar zcvf db.tar.gz db.pgdump
```

To unpack the tar file:

```
tar zxvf db.tar.gz
```

Run reinit-db-instance to make the changes take effect.

10 Create a new DHIS2 Live package on dhis2.org and place it in download/live directory. Only the WAR file must be updated. An uncompressed Live package is located on the demo server at:

```
/home/dhis/dhis-live-package
```

Replace the uncompressed WAR file with the new release. Make a compressed Live archive and move to /download/live directory.

11 Upload sample database to dhis2.org and place it in download/resources directory.

12 Update download page at www.dhis2.org/downloads with links to new Live package, WAR file, source code branch page and sample data including version.

13 Add stable WAR file download link in nginx.conf on dhis2.org.

14 Create a documentation branch and update documentation page at www.dhis2.org/documentation with links to documentation and Javadocs.

15 Write and send release email.

Chapter 4. DHIS2 Technical Architecture

4.1. Overview

This document outlines the technical architecture for the District Health Information Software 2 (DHIS2). The DHIS2 is a routine data based health information system which allows for data capture, aggregation, analysis, and reporting of data.

DHIS2 is written in Java and has a three-layer architecture. The presentation layer is web-based, and the system can be used on-line as well as stand-alone.
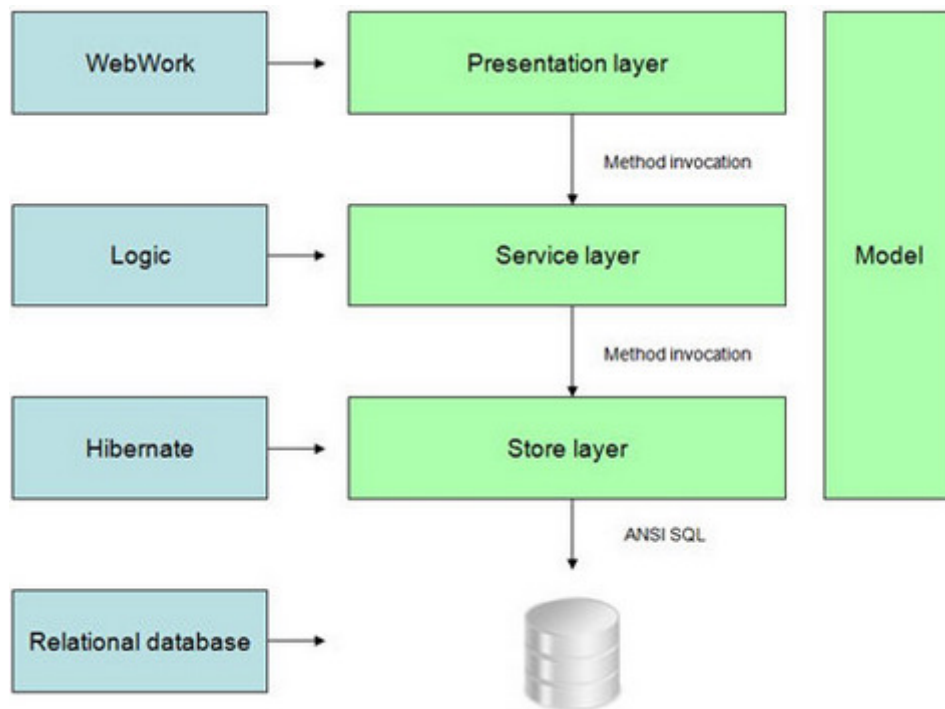

Fig. Overall architecture

4.2. Technical Requirements

The DHIS2 is intended to be installed and run on a variety of platforms. Hence the system is designed for industry standards regarding database management systems and application servers. The system should be extensible and modular in order to allow for third-party and peripheral development efforts. Hence a pluggable architecture is needed. The technical requirements are:

- Ability to run on any major database management system
- Ability to run on any J2EE compatible servlet container
- Extensibility and modularity in order to address local functional requirements
- Ability to run on-line/on the web
- Flexible data model to allow for a variety of data capture requirements

4.3. Project Structure

DHIS2 is made up of 42 Maven projects, out of which 18 are web modules. The root POM is located in `/dhis-2` and contains project aggregation for all projects excluding the `/dhis-2/dhis-`

web folder. The `/dhis-2/dhis-web` folder has a web root POM which contains project aggregation for all projects within that folder. The contents of the modules are described later on.
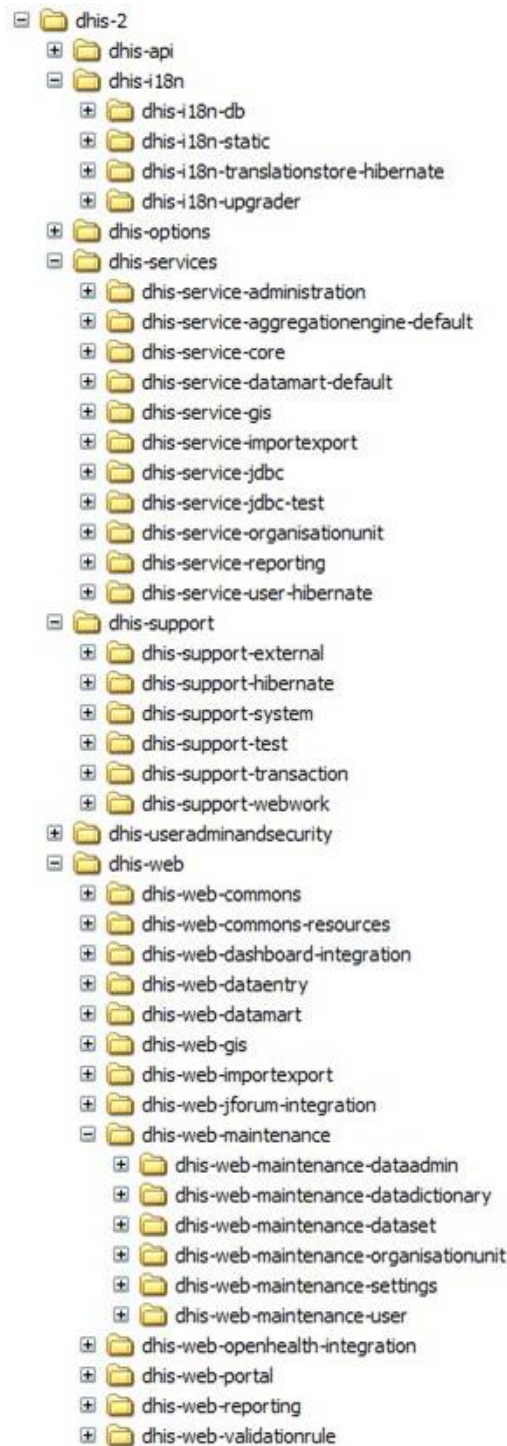
Fig. Project structure

4.4. Project Dependencies

Dependencies between the projects are structured in five layers. The support modules provide support functionality for the core and service modules, related to Hibernate, testing, JDBC, and the file system. The core module provides the core functionality in the system, like persistence and business logic for the central domain objects. The service modules provide business logic for services related to reporting, import-export, mapping, and administration. The web modules are self-contained web modules. The portal is a wrapper web module which assembles all the

web modules. Modules from each layer can only have dependencies to modules at the same layer or the layer right below.
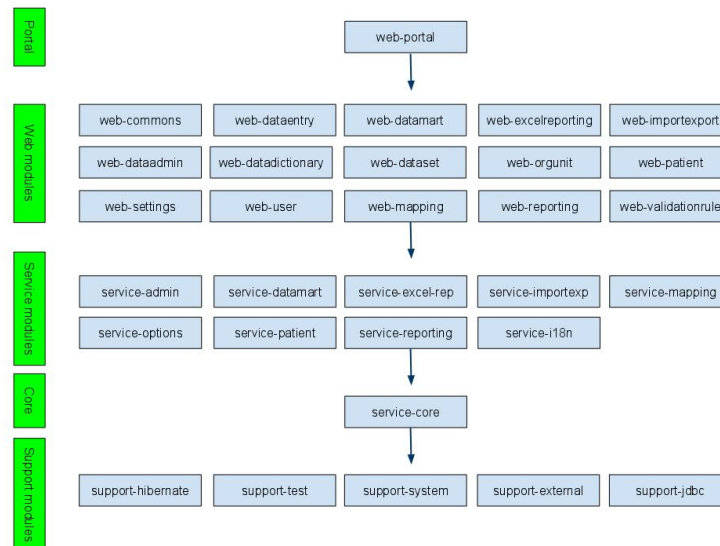
The internal structure of the service layer is divided in five layers.
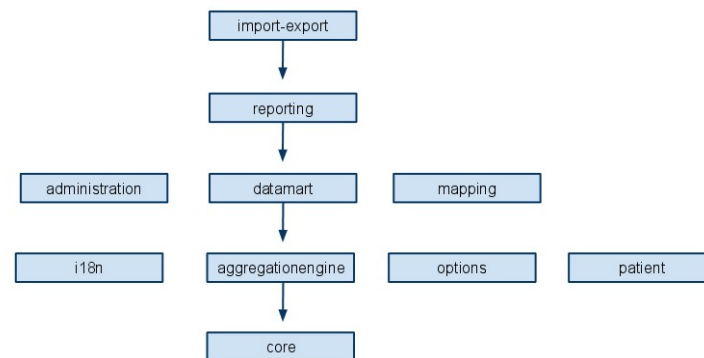
4.5. The Data Model

The data model is flexible in all dimensions in order to allow for capture of any item of data. The model is based on the notion of a DataValue. A DataValue can be captured for any DataElement (which represents the captured item, occurrence or phenomena), Period (which represents the time dimension), and Source (which represents the space dimension, i.e. an organisational unit in a hierarchy).
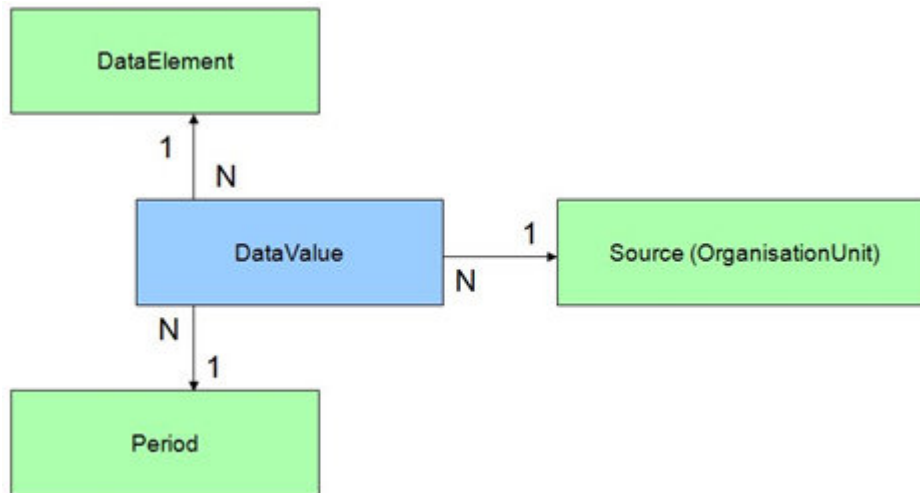
Figure 4.1. Data values

A central concept for data capture is the DataSet. The DataSet is a collection of DataElements for which there is entered data presented as a list, a grid and a custom designed form. A DataSet is associated with a PeriodType, which represents the frequency of data capture.

A central concept for data analysis and reporting is the Indicator. An Indicator is basically a mathematical formula consisting of DataElements and numbers. An Indicator is associated with an IndicatorType, which indicates the factor of which the output should be multiplied with. A typical IndicatorType is percentage, which means the output should be multiplied by 100. The formula is split into a numerator and denominator.

Most objects have corresponding group objects, which are intended to improve and enhance data analysis. The data model source code can be found in the API project and could be explored in entirety there. A selection of the most important objects can be view in the diagram below.
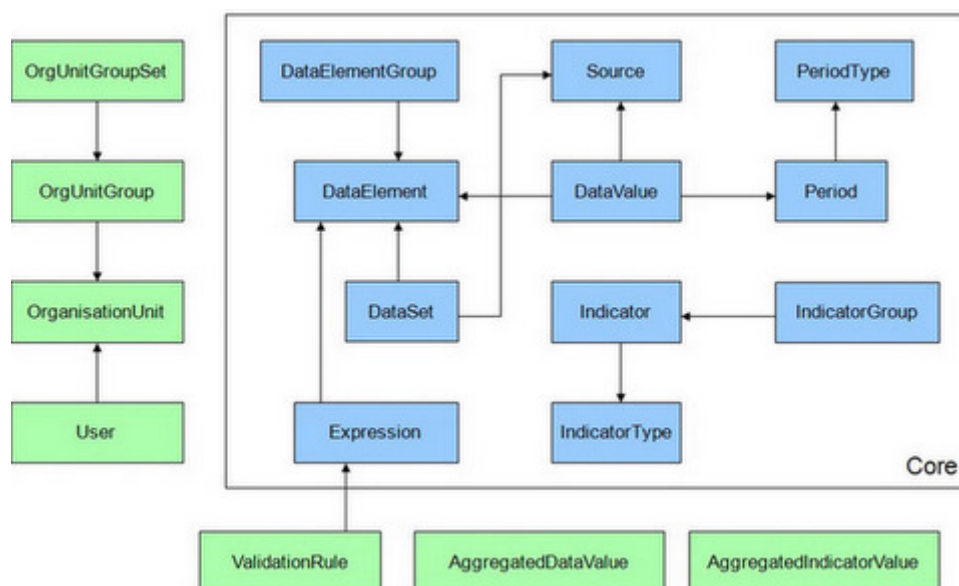
Figure 4.2. Core Diagram

4.6. The Persistence Layer

The persistence layer is based on Hibernate in order to achieve the ability to run on any major DBMS. Hibernate abstracts the underlying DBMS away and let you define the database connection properties in a file called hibernate.properties.

DHIS2 uses Spring-Hibernate integration, and retrieves a SessionFactory through Spring's LocalSessionFactoryBean. This LocalSessionFactoryBean is injected with a custom HibernateConfigurationProvider instance which fetches Hibernate mapping files from all modules currently on the classpath. All store implementations get injected with a SessionFactory and use this to perform persistence operations.

Most important objects have their corresponding Hibernate store implementation. A store provides methods for CRUD operations and queries for that object, e.g. HibernateDataElementStore which offers methods such as addDataElement(DataElement), deleteDataElement(DataElement), getDataElementByName(String), etc.
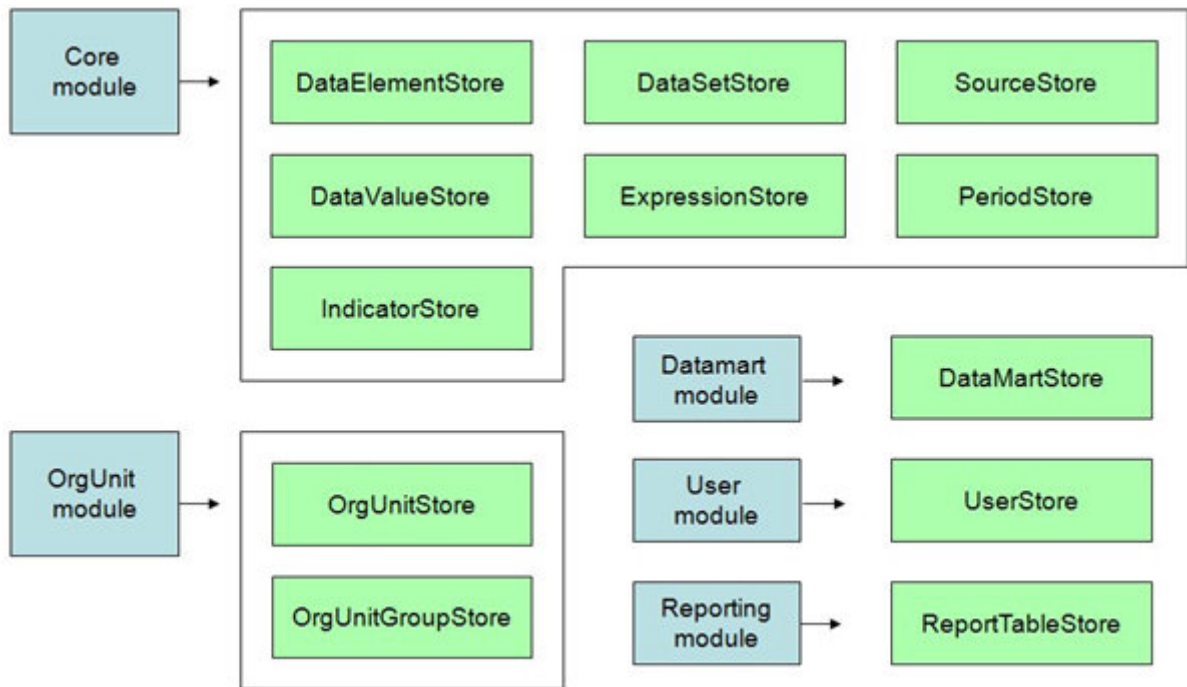
Fig. Persistence layer

4.7. The Business Layer

All major classes, like those responsible for persistence, business logic, and presentation, are mapped as Spring managed beans. "Bean" is Spring terminology and simply refers to a class that is instantiated, assembled, and otherwise managed by the Spring IoC container. Dependencies between beans are injected by the IoC container, which allows for loose coupling, re-configuration and testability. For documentation on Spring, please refer to springframework.org.

The services found in the dhis-service-core project basically provide methods that delegate to a corresponding method in the persistence layer, or contain simple and self-explanatory logic. Some services, like the ones found in the dhis-service-datamart, dhis-service-import-export, dhis-service-jdbc, and dhis-service-reporting projects are more complex and will be elaborated in the following sections.

4.7.1. The JDBC Service Project

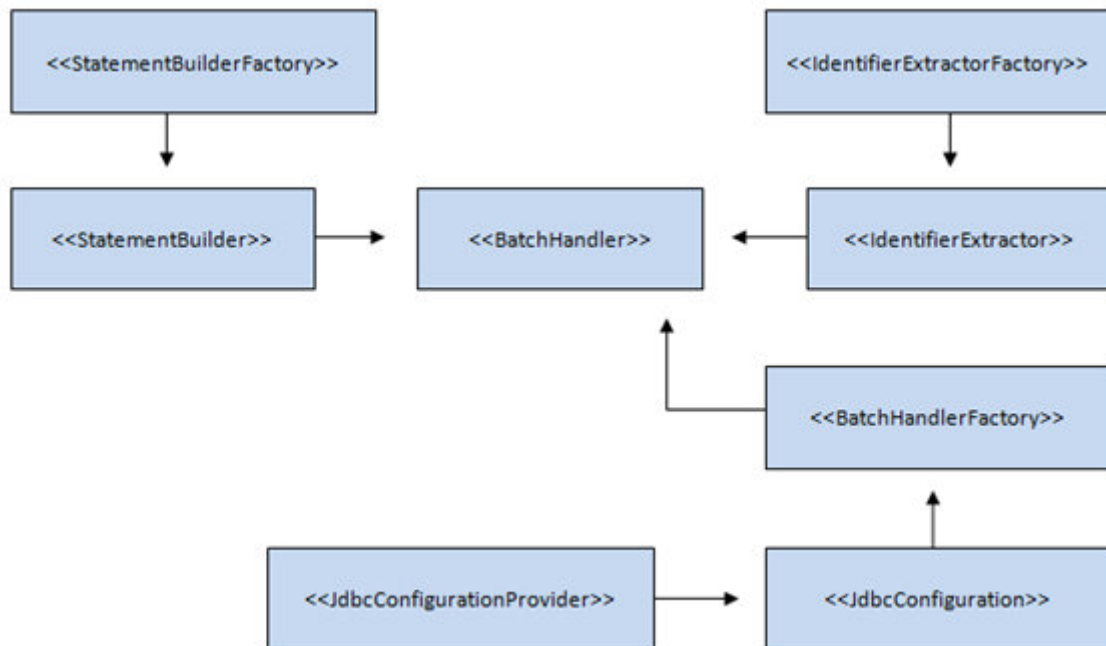The JDBC service project contains a set of components dealing with JDBC connections and SQL statements.

Fig. JDBC BatchHandler diagram

The BatchHandler interface provides methods for inserting, updating and verifying the existence of objects. The purpose is to provide high-performance operations and is relevant for large amounts of data. The BatchHandler object inserts objects using the multiple insert SQL syntax behind the scenes and can insert thousands of objects on each database commit. A typical use-case is an import process where a class using the BatchHandler interface will call the addObject(Object, bool) method for every import object. The BatchHandler will after an appropriate number of added objects commit to the database transparently. A BatchHandler can be obtained from the BatchHandlerFactory component. BatchHandler implementations exist for most objects in the API.

The JdbcConfiguration interface holds information about the current DBMS JDBC configuration, more specifically dialect, driver class, connection URL, username and password. A JdbcConfiguration object is obtained from the JdbcConfigurationProvider component, which currently uses the internal Hibernate configuration provider to derive the information.

The StatementBuilder interface provides methods that represents SQL statements. A StatementBuilder object is obtained from the StatementBuilderFactory, which is able to determine the current runtime DBMS and provide an appropriate implementation. Currently implementations exist for PostgreSQL, MySQL, H2, and Derby.

The IdentifierExtractor interface provides methods for retrieving the last generated identifiers from the DBMS. An IdentifierExtractor is obtained from the IdentifierExtractorFactory, which is able to determine the runtime DBMS and provide an appropriate implementation.
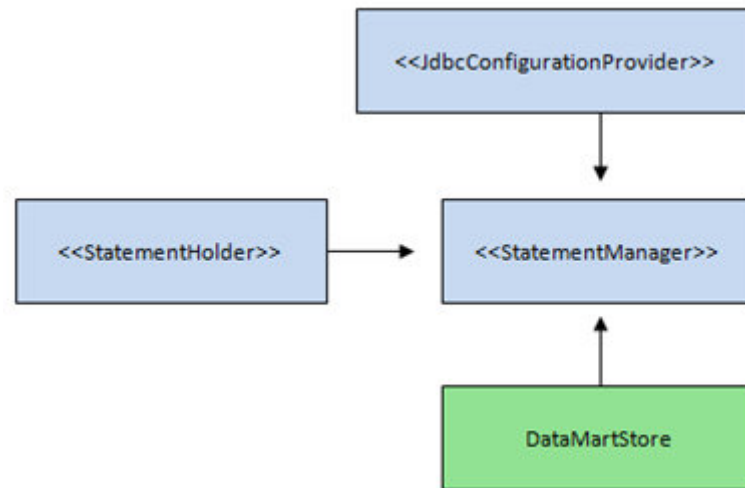
Fig. JDBC StatementManager diagram

The StatementHolder interface holds and provides JDBC connections and statements. A StatementHolder object can be obtained from the StatementManager component. The StatementManager can be initialized using the initalise() method closed using the destroy() method. When initialized, the StatementManager will open a database connection and hold it in a ThreadLocal variable, implying that all subsequent requests for a StatementHolder will return the same instance. This can be used to improve performance since a database connection or statement can be reused for multiple operations. The StatementManager is typically used in the persistence layer for classes working directly with JDBC, like the DataMartStore.

4.7.2. The Reporting Project

The reporting project contains components related to reporting, which will be described in the following sections.

4.7.2.1. Report table

The ReportTable object represents a crosstabulated database table. The table can be crosstabulated on any number of its three dimensions, which are the descriptive dimension (which can hold data elements, indicators, or data set completeness), period dimension, and organisation unit dimension. The purpose is to be able to customize tables for later use either in third-party reporting tools like BIRT or directly in output formats like PDF or HTML inside the system. Most of the logic related to crosstabulation is located in the ReportTable object. A ReportTable can hold:

- Any number of data elements, indicators, data sets, periods, and organisation units.

- A RelativePeriods object, which holds 10 variants of relative periods. Examples of such periods are last 3 months, so far this year, and last 3 to 6 months. These periods are relative to the reporting month. The purpose of this is to make the report table re-usable in time, i.e. avoid the need for the user to replace periods in the report table as time goes by.

- A ReportParams object, which holds report table parameters for reporting month, parent organisation unit, and current organisation unit. The purpose is to make the report table re-usable across the organisation unit hierarchy and in time, i.e. make it possible for the user to re-use the report table across organisation units and as time goes by.

- User options such as regression lines. Value series which represents regression values can be included when the report table is crosstabulated on the period dimension.
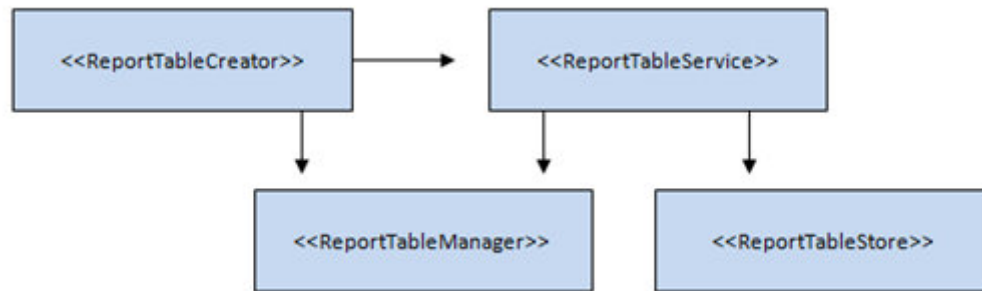
Fig. Report table diagram

The ReportTableStore is responsible for persisting ReportTable objects, and currently has a Hibernate implementation.

The ReportTableService is responsible for performing business logic related to report tables such as generation of relative periods, as well as delegating CRUD operations to the ReportTableStore.

The ReportTableManager is responsible for creating and removing report tables, as well as retrieving data.

The ReportTableCreator is the key component, and is responsible for:

- Exporting relevant data to the data mart using the DataMartExportService or the DataSetCompletenessService. Data will later be retrieved from here and used to populate the report table.
- Create the report table using the ReportTableManager.
- Include potential regression values.
- Populate the report table using a BatchHandler.
- Remove the report table using the ReportTableManager.

4.7.2.2. Chart
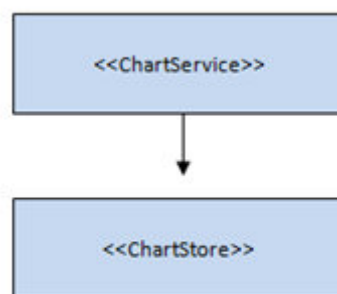
The Chart object represents preferences for charts. Charts are either period based or organisation unit based. A chart has tree dimensions, namely the value, category, and filter dimension. The value dimension contains any numbers of indicators. In the period based chart, the category dimension contains any number of periods while the filter dimension contains a single organisation unit. In the organisation unit based chart, the category dimension contains any number of organisation units while the filter dimension contains a single period. Two types of charts are available, namely bar charts and line charts. Charts are materialized using the JFreeChart library. The bar charts are rendered with a BarRenderer [2], the line charts with a LineAndShapeRenderer [2], while the data source for both variants is a DefaultCategoryDataSet [3]. The ChartService is responsible for CRUD operations, while the ChartService is responsible for creating JfreeCharts instances based on a Chart object.

Fig. Chart diagram

4.7.2.3. Data set completeness

The purpose of the data set completeness functionality is to record the number of data sets that have been completed. The definition of when a data set is complete is subjective and based on a function in the data entry screen where the user can mark the current data set as complete. This functionality provides a percentage completeness value based on the number of reporting organisation units with completed data sets compared to the total number of reporting organisation units for a given data set. This functionality also provides the number of completed data sets reported on-time, more specifically reported before a defined number of days after the end of the reporting period. This date is configurable.
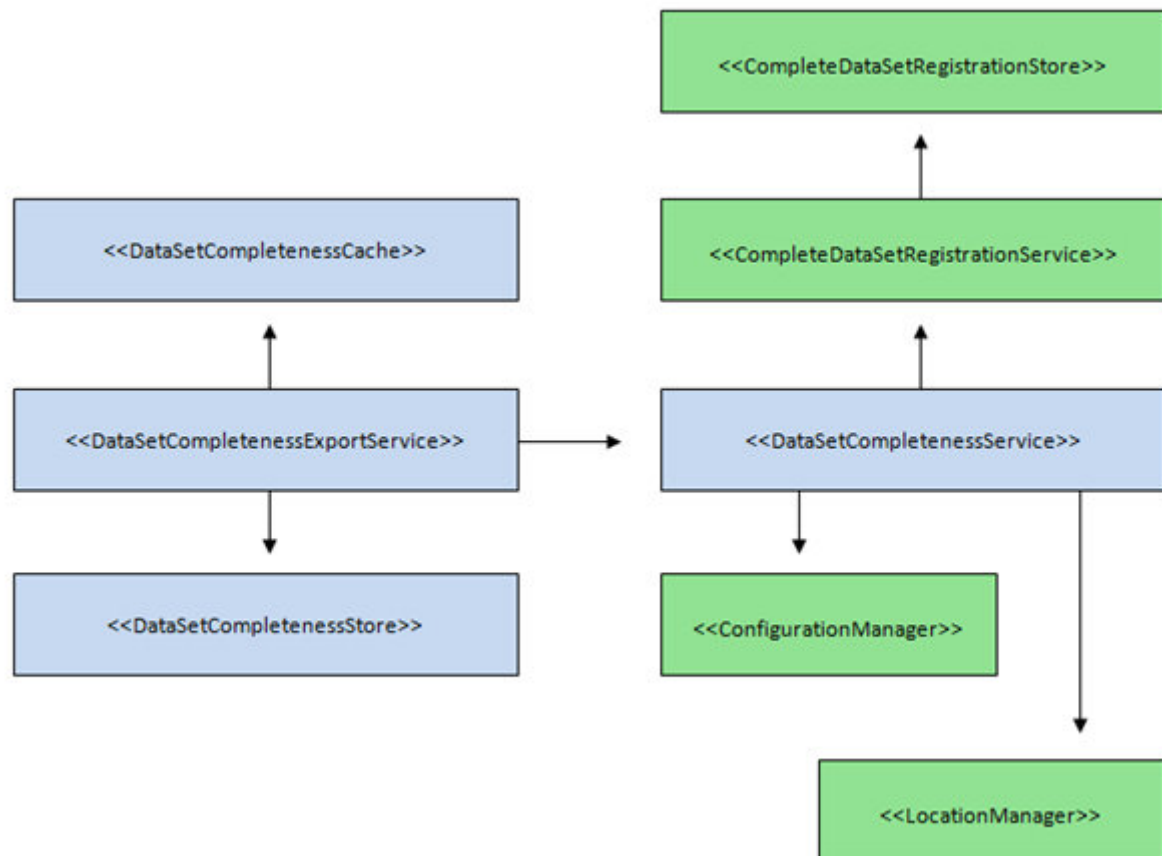

Fig. Data set completeness diagram

The CompleteDataSetRegistration object is representing a data set marked as complete by a user. This property holds the data set, period, organisation unit and date for when the complete registrations took place. The CompleteDataSetRegistrationStore is responsible for persistence of CompleteDataSetRegistration objects and provides methods returning collections of objects queried with different variants of data sets, periods, and organisation units as input parameters. The CompleteDataSetRegistrationService is mainly delegating method calls the store layer. These components are located in the dhis-service-core project.

The completeness output is represented by the DataSetCompletenessResult object. This object holds information about the request that produced it such as data set, period, organisation unit, and information about the data set completeness situation such as number of reporting organisation units, number of complete registrations, and number of complete registrations on-time. The DataSetCompletenessService is responsible for the business logic related to data set completeness reporting. It provides methods which mainly returns collections of DataSetCompletenessResults and takes different variants of period, organisation unit and data set as parameters. It uses the CompleteDataSetRegistrationService to retrieve the number

of registrations, the DataSetService to retrieve the number of reporting organisation units, and performs calculations to derive the completeness percentage based on these retrieved numbers.

The DataSetCompletenessExportService is responsible for writing DataSetCompletenessResults to a database table called "aggregateddatasetcompleteness". This functionality is considered to be part of the data mart as this data can be used both inside DHIS2 for e.g. report tables and by third-party reporting applications like MS Excel. This component is retrieving data set completeness information from the DataSetCompeletenessService and is using the BatchHandler interface to write such data to the database.

4.7.2.4. Document

The Document object represents either a document which is uploaded to the system or a URL. The DocumentStore is responsible for persisting Document objects, while the DocumentService is responsible for business logic.
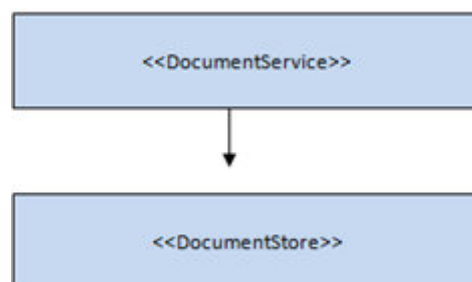

Fig. Document diagram

4.7.2.5. Pivot table

The PivotTable object represents a pivot table. It can hold any number of indicators, periods, organisation units, and corresponding aggregated indicator values. It offers basic pivot functionality like pivoting and filtering the table on all dimensions. The business logic related to pivot tables is implemented in Javascript and is located in the presentation layer. The PivotTableService is reponsible for creating and populating PivotTable objects.

4.7.3. The External Project

The LocationManager component is responsible for the communication between DHIS2 and the file system of the operating system. It contains methods which provide read access to files through File and InputStream instances, and write access to the file system through File and OutputStream instances. The target location is relative to a system property "dhis2.home" and an environment variable "DHIS2_HOME" in that order. This component is used e.g. by the HibernateConfigurationProvider to read in the Hibernate configuration file, and should be re-used by all new development efforts.

The ConfigurationManager is a component which facilitates the use of configuration files for different purposes in DHIS2. It provides methods for writing and reading configuration objects to and from XML. The XStream library is used to implement this functionality. This component is typically used in conjunction with the LocationManager.

4.7.4. The System Support Project

The system support project contains supportive classes that are general and can be reused througout the system.

4.7.4.1. DeletionManager

The deletion manager solution is responsible for deletion of associated objects. When an object has a depdency to another object this association needs to be removed by the application before the latter object can be deleted (unless the association is defined to be cascading in the DBMS). Often an object in a peripheral module will have an associations to a core object. When deleting the core object this association must be removed before deleting the core object.The core module cannot have a dependency to the peripheral module however due to the system design and the problem of cyclic dependencies. The deletion manager solves this by letting all objects implement a DeletionHandler which takes care of associations to other objects. A DeletionHandler should override methods for objects that, when deleted, will affect the current object in any way. The DeletionHandler can choose to disallow the deletion completely by overriding the allowDelete* method, or choose to allow the deletion and remove the associations by overriding the delete* method. Eg. a DeletionHandler for DataElementGroup should override the deleteDataElement(..) method which should remove the DataElement from all DataElementGroups. If one decide that DataElement which are a member of any DataElementGroups cannot be deleted, it should override the allowDeleteDataElement() method and return false if there exists DataElementGroups with associations to that DataElement.

First, all DeletionHandler implementations are registered with the DeletionManager through a Spring MethodInvokingFactoryBean in the Spring config file. This solution adheres to the observer design pattern.

Second, all method invocations that should make the DeletionManager execute are mapped to the DeletionInterceptor with Spring AOP advice in the Spring config file. The DeletionInterceptor in turn invokes the execute method of the DeletionManager. First, the DeletionManager will through reflection invoke the allowDelete* method on all DeletionHandlers. If no DeletionHandlers returned false it will proceed to invoke the delete* method on all DeletionHandlers. This way all DeletionHandlers get a chance to clean up associations to the object being deleted. Finally the object itself is deleted.

4.8. The Presentation Layer

The presentation layer of DHIS2 is based on web modules which are assembled into a portal. This implies a modularized design where each module has its own domain, e.g. the dhis-web-reporting module deals with reports, charts, pivot tables, documents, while the dhis-web-maintenance-dataset module is responsible for data set management. The web modules are based on Struts and follow the MVC pattern [5]. The modules also follow the Maven standard for directory layout, which implies that Java classes are located in src/main/java, configuration files and other resources in src/main/resources, and templates and other web resources in src/main/webapp. All modules can be run as a standalone application.

Common Java classes, configuration files, and property files are located in the dhis-web-commons project, which is packaged as a JAR file. Common templates, style sheets and other web resources are located in the dhis-web-commons-resources project, which is packaged as a WAR file. These are closely related but are separated into two projects. The reason for this is that other modules must be able to have compile dependencies on the common Java code, which requires it to be packaged as a JAR file. For other modules to be able to access the common web resources, these must be packaged as a WAR file [6].

4.8.1. The Portal

DHIS2 uses a light-weight portal construct to assemble all web modules into one application. The portal functionality is located in the dhis-web-portal project. The portal solution is

integrated with Struts, and the following section requires some prior knowledge about this framework, please refer to struts.apache.org for more information.

4.8.1.1. Module Assembly

All web modules are packaged as WAR files. The portal uses the Maven WAR plug-in to assemble the common web modules and all web modules into a single WAR file. Which modules are included in the portal can be controlled simply through the dependency section in the POM file [7] in the dhis-web-portal project. The web module WAR files will be extracted and its content merged together.

4.8.1.2. Portal Module Requirements

The portal requires the web modules to adhere to a few principles:

- The web resources must be located in a folder `src/main/webapp/<module-artifact-id >`.
- The xwork.xml configuration file must extend the dhis-web-commons.xml configuration file.
- The action definitions in xwork.xml for a module must be in a package where the name is <module-artifact-id>, namespace is /<module-artifact-id>, and which extends the dhis-web-commons package.
- All modules must define a default action called index.
- The web.xml of the module must define a redirect filter, open-session-in-view filter, security filter, and the Struts FilterDispatcher [8].
- All modules must have dependencies to the dhis-web-commons and dhis-web-commons-resources projects.

4.8.1.3. Common Look-And-Feel

Common look and feel is achieved using a back-bone Velocity template which includes a page template and a menu template defined by individual actions in the web modules. This is done by using static parameters in the Struts/Xwork xwork.xml configuration file. The action response is mapped to the back-bone template called main.vm, while static parameters called page and menu refers to the templates that should be included. This allows the web modules to display its desired content and left side menu while maintaining a common look-and-feel.

4.8.1.4. Main Menu

The main menu contains links to each module. Each menu link will redirect to the index action of each module. The menu is updated dynamically according to which web modules are on the classpath. The menu is visibly generated using the ModuleManager component, which provides information about which modules are currently included. A module is represented by the Module object, which holds properties about the name, package name, and default action name. The ModuleManager detects web modules by reading the Struts Configuration and PackageConfig objects, and derives the various module names from the name of each package definition. The Module objects are loaded onto the Struts value stack by Struts interceptors using the ModuleManager. These values are finally used in the back-bone Velocity template to produce the menu mark-up.

4.9. Definitions

[1] "Classpath" refers to the root of a JAR file, /WEB-INF/lib or /WEB-INF/classes in a WAR-file and /src/main/resources in the source code; locations from where the JVM is able to load classes.

[2] JFreeChart class located in the org.jfree.chart.renderer package.

[3] JFreeChart class located in the org.jfree.data.category package.

[4] Operations related to creating, retrieving, updating, and deleting objects.

[5] Model-View-Controller, design pattern for web applications which separates mark-up code from application logic code.

[6] The WAR-file dependency is a Maven construct and allows projects to access the WAR file contents during runtime.

[7] Project Object Model, the key configuration file in a Maven 2 project.

[8] Represents the front controller in the MVC design pattern in Struts.

[9] Hibernate second-level cache does not provide satisfactory performance.

Appendix A. DHIS2 and R integration

A.1. Introduction

R is freely available, open source statistical computing environment. R refers to both the computer programming language, as well as the software which can be used to create and run R scripts. There are [numerous sources on the web](#) which describe the extensive set of features of R.

R is a natural extension to DHIS2, as it provides powerful statistical routines, data manipulation functions, and visualization tools. This chapter will describe how to setup R and DHIS2 on the same server, and will provide a simple example of how to retrieve data from the DHIS2 database into an R data frame and perform some basic calculations.

A.2. Installing R

If you are installing R on the same server as DHIS, you should consider using the Comprehensive R Archive Network (CRAN) to get the latest distribution of R. All you need to do is to add the following like to you `/etc/apt/source.list` file.

deb <your R mirror>/bin/linux/ubuntu <your Ubuntu distribution>

You will need to replace <your R mirror> with one from the list available [here.](#) You will also need to replace <your Ubuntu distribution> with the name of the distribution you are using.

Once you have done this, invoke the following commands

```
sudo apt-get update
gpg --keyserver pgp.mit.edu --recv-keys 51716619E084DAB9
gpg --armor --export 51716619E084DAB9 | apt-key add -
sudo apt-get install r-base r-cran-dbi
```

At this point, you should have a functional R installation on your machine.

Next, lets see if everything is working by simpling invoking R from the command line.

```
foo@bar:~$ R

R version 2.14.1 (2011-12-22)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i686-pc-linux-gnu (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

A.3. Using ODBC to retrieve data from DHIS2 into R

In this example, we will use a system-wide ODBC connector which will be used to retrieve data from the DHIS2 database. There are some disadvantages with this approach, as ODBC is slower than other methods and it does raise some security concerns by providing a system-wide connector to all users. However, it is a convenient method to provide a connection to multiple users. The use of the R package RODBC will be used in this case. Other alternatives would be the use of the [RPostgreSQL](#) package, which can interface directly through the PostgreSQL driver described in the next section.

Assuming you have already installed R from the procedure in the previous section. Invoke the following command to add the required libraries for this example.

apt-get install r-cran-rodbc r-cran-lattice odbc-postgresql

Next, we need to configure the ODBC connection. Edit the file to suit your local situation using the following template as a guide. Lets create and edit a file called odbc.ini

```
[dhis2]
Description          = DHIS2 Database
Driver               = /usr/lib/odbc/psqlodbcw.so
Trace                = No
TraceFile            = /tmp/sql.log
Database             = dhis2
Servername           = 127.0.0.1
UserName             = postgres
Password             = SomethingSecure
Port                 = 5432
Protocol             = 9.0
ReadOnly             = Yes
RowVersioning        = No
ShowSystemTables     = No
ShowOidColumn        = No
FakeOidIndex         = No
ConnSettings         =
Debug = 0
```

Finally, we need to install the ODBC connection with odbcinst -i -d -f odbc.ini

From the R prompt, execute the following commands to connect to the DHIS2 database.

```
> library(RODBC)
> channel<-odbcConnect("dhis2")#Note that the name must match the ODBC connector name
> sqlTest<-c("SELECT dataelementid, name FROM dataelement LIMIT 10;")
> sqlQuery(channel,sqlTest)
                                                                      name
1   OPD First Attendances Under 5
2   OPD First Attendances Over 5
3   Deaths Anaemia Under 5 Years
4   Deaths Clinical Case of Malaria Under 5 Years
5   Inpatient discharges under 5
6   Inpatient Under 5 Admissions
7   Number ITNs
8   OPD 1st Attendance Clinical Case of Malaria Under 5
9  IP Discharge Clinical Case of Malaria Under 5 Years
10 Deaths of malaria case provided with anti-malarial treatment 1 to 5 Years
>
```

It seems R is able to retrieve data from the DHIS2 database.

As an illustrative example, lets say we have been asked to calculate the relative percentage of OPD male and female under 5 attendances for the last twelve months.First, lets create an SQL query which will provide us the basic information which will be required.

```
OPD<-sqlQuery(channel,"SELECT p.startdate, de.name as de, sum(dv.value::double
 precision)
 FROM datavalue dv
 INNER JOIN period p on dv.periodid = p.periodid
 INNER JOIN dataelement de on dv.dataelementid = de.dataelementid
 WHERE p.startdate >= '2011-01-01'
 and p.enddate <= '2011-12-31'
 and de.name ~*('Attendance OPD')
 GROUP BY p.startdate, de.name;")
```

We have stored the result of the SQL query in an R data frame called "OPD". Lets take a look
at what the data looks like.

```
> head(OPD)
   startdate                                de     sum
1 2011-12-01   Attendance OPD <12 months female  42557
2 2011-02-01   Attendance OPD <12 months female 127485
3 2011-01-01   Attendance OPD 12-59 months male 200734
4 2011-04-01   Attendance OPD 12-59 months male 222649
5 2011-06-01   Attendance OPD 12-59 months male 168896
6 2011-03-01 Attendance OPD 12-59 months female 268141
> unique(OPD$de)
[1] Attendance OPD <12 months female   Attendance OPD 12-59 months male
[3] Attendance OPD 12-59 months female Attendance OPD >5 years male
[5] Attendance OPD <12 months male     Attendance OPD >5 years female
6 Levels: Attendance OPD 12-59 months female ... Attendance OPD >5 years male
>
```

We can see that we need to aggregate the two age groups (< 12 months and 12-59 months)
into a single variable, based on the gender. Lets reshape the data into a crosstabulated table
to make this easier to visualize and calculate the summaries.

```
>OPD.ct<-cast(OPD,startdate ~ de)
>colnames(OPD.ct)
[1] "startdate"                          "Attendance OPD 12-59 months female"
[3] "Attendance OPD 12-59 months male"   "Attendance OPD <12 months female"
[5] "Attendance OPD <12 months male"     "Attendance OPD >5 years female"
[7] "Attendance OPD >5 years male"
```

We have reshaped the data so that the data elements are individual columns. It looks like
we need to aggregate the second and fourth columns together to get the under 5 female
attendance, and then the third and fifth columns to get the male under 5 attendance.After this,
lets subset the data into a new data frame just to get the required information and display
the results.

```
> OPD.ct$OPDUnder5Female<-OPD.ct[,2]+OPD.ct[,4]#Females
> OPD.ct$OPDUnder5Male<-OPD.ct[,3]+OPD.ct[,5]#males
> OPD.ct.summary<-OPD.ct[,c(1,8,9)]#new summary data frame
>OPD.ct.summary$FemalePercent<-
OPD.ct.summary$OPDUnder5Female/
(OPD.ct.summary$OPDUnder5Female + OPD.ct.summary$OPDUnder5Male)*100#Females
>OPD.ct.summary$MalePercent<-
OPD.ct.summary$OPDUnder5Male/
(OPD.ct.summary$OPDUnder5Female + OPD.ct.summary$OPDUnder5Male)*100#Males
```

Of course, this could be accomplished much more elegantly, but for the purpose of the
illustration, this code is rather verbose.Finally, lets display the required information.

```
> OPD.ct.summary[,c(1,4,5)]
    startdate FemalePercent MalePercent
1  2011-01-01      51.13360     48.86640
```

```
2  2011-02-01      51.49154      48.50846
3  2011-03-01      51.55651      48.44349
4  2011-04-01      51.19867      48.80133
5  2011-05-01      51.29902      48.70098
6  2011-06-01      51.66519      48.33481
7  2011-07-01      51.68762      48.31238
8  2011-08-01      51.49467      48.50533
9  2011-09-01      51.20394      48.79606
10 2011-10-01      51.34465      48.65535
11 2011-11-01      51.42526      48.57474
12 2011-12-01      50.68933      49.31067
```

We can see that the male and female attendances are very similar for each month of the year, with seemingly higher male attendance relative to female attendance in the month of December.

In this example, we showed how to retrieve data from the DHIS2 database and manipulate in with some simple R commands. The basic pattern for using DHIS2 and R together, will be the retrieval of data from the DHIS2 database with an SQL query into an R data frame, followed by whatever routines (statistical analysis, plotting, etc) which may be required.

A.4. Using R with MyDatamart

MyDatamart provides useful interface to the DHIS2 database by making a local copy of the database available on a users desktop. This means that the user does not need direct access to the database and the data can be worked with offline on the users local machine. In this example, we will have used the demo database. Data was downloaded at the district level for Jan 2011-Dec 201l. Consult the MyDatamart section in this manual for more detailed information.

First, lets load some required R packages. If you do not have these packages already installed in your version of R, you will need to do so before proceeding with the example.

```
library("DBI")
library("RSQLite")
library("lattice")
library("latticeExtra")
```

Next, we are going to connect to the local copy of the MyDatamart database. In this case, it was located at C:\dhis2\sl.dmart.

```
dbPath<-"C:\\dhis2\\sl.dmart"
drv<-dbDriver("SQLite")
db<-dbConnect(drv,dbPath)
```

Let suppose we have been asked to compare ANC 1, 2, 3 coverage rates for each district for 2011. We can define an SQL query to retrieve data from the MyDatamart database into an R data frame as follows.

```
#An SQL query which will retreive all indicators
#at OU2 le
sql<-"SELECT * FROM pivotsource_indicator_ou2_m
WHERE year = '2011'"
#Execute the query into a new result set
rs<-dbSendQuery(db,sql)
#Put the entire result set into a new data frame
Inds<-fetch(rs,n=-1)
#Clean up a bit
dbClearResult(rs)
dbDisconnect(db)
```

We used one of the pre-existing Pivot Source queries in the database to get all of the indicator values. Of course, we could have retrieved only the ANC indicators, but we did not exactly know how the data was structured, or how the columns were named, so lets take a closer look.

```
#Get the name of the columns
colnames(Inds)
#output not shown for brevity
levels(as.factor(Inds$indshort))
```

We see from the colnames command that there is an column called "indshort" which looks like it contains some indicator names. We can see the names using the second command. After we have determined which ones we need (ANC 1, 2, and 3), lets further subset the data so that we only have these.

```
#Subset the data for ANC
ANC<-Inds[grep("ANC (1|2|3) Coverage",as.factor(Inds$indshort)),]
```

We just used R's grep function to retrieve all the rows and columns of the Inds data frame which matched the regular expression "ANC (1|2|3) Coverage" and put this into a new data frame called "ANC".

By looking at the data with the str(ANC) command, we will notice that the time periods are not ordered correctly, so lets fix this before we try and create a plot of the data.

```
#Lets reorder the months
MonthOrder<-c('Jan','Feb','Mar','Apr',
'May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')
ANC$month<-factor(ANC$month,levels=MonthOrder)
```

Next, we need to actually calculate the indicator value from the numerator, factor and denominator.

```
#Calculate the indicator value
ANC$value<-ANC$numxfactor/ANC$denominatorvalue
```

Finally, lets create a simple trellis plot which compares ANC 1, 2, 3 for each district by month and save it to our local working directory in a file called "District_ANC.png".

```
png(filename="District_ANC.png",width=1024,height=768)
plot.new()
 xyplot(value ~ month | ou2, data=ANC, type="a", main="District ANC Comparison Sierra
 Leone 2011",
 groups=indshort,xlab="Month",ylab="ANC Coverage",
 scales = list(x = list(rot=90)),
 key = simpleKey(levels(factor(ANC$indshort)),
 points=FALSE,lines=TRUE,corner=c(1,1)))
 mtext(date(), side=1, line=3, outer=F, adj=0, cex=0.7)
dev.off()
```

The results of which are displayed below.
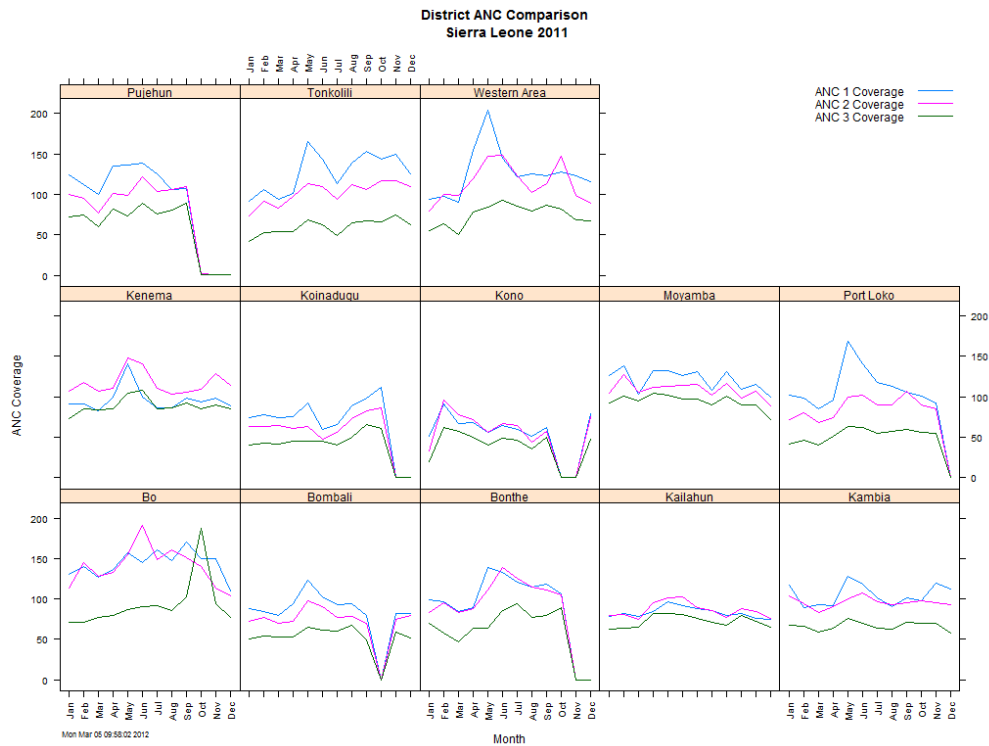
A.5. Mapping with R and PostgreSQL

A somewhat more extended example, will use the RPostgreSQL library and several other libraries to produce a map from the coordinates stored in the database. We will define a few helper functions to provide a layer of abstraction, which will make the R code more reusable.

```
#load some dependent libraries
 library(maps)
 library(maptools)
 library(ColorBrewer)
 library(ClassInt)
 library(RPostgreSQL)

#Define some helper functions

#Returns a dataframe from the connection for a valid statement
dfFromSQL<-function (con,sql){
    rs<-dbSendQuery(con,sql)
    result<-fetch(rs,n=-1)
    return(result)
}
#Returns a list of latitudes and
 longitudes from the orgunit table
dhisGetFacilityCoordinates<- function(con,levelLimit=4) {
sqlCoords<-paste("SELECT ou.organisationunitid, ou.name,
substring(ou.coordinates from E'(?=,?)-[0-9]+\\.[0-9]+')::double precision as
 latitude,
substring(ou.coordinates from E'[0-9\\.]+'):double precision as
 longitude FROM organisationunit ou where ou.organisationunitid
 in (SELECT DISTINCT idlevel",levelLimit, " from _orgunitstructure)
 and ou.featuretype = 'Point'
 ;",sep="")
 result<-dfFromSQL(con,sqlCoords)
 return(result)
 }
```

```
#Gets a dataframe of IndicatorValues,
# provided the name of the indicator,
# startdate, periodtype and level
dhisGetAggregatedIndicatorValues<-function(con,
indicatorName,
startdate,
periodtype="Yearly",
level=4)
{
  sql<-paste("SELECT organisationunitid,dv.value FROM aggregatedindicatorvalue dv
where dv.indicatorid  =
(SELECT indicatorid from indicator where name = \'",indicatorName,"\') and dv.level
 =", level,"and
 dv.periodid  =
(SELECT periodid from period where
startdate = \'",startdate,"\'
and periodtypeid =
(SELECT periodtypeid from periodtype
 where name = \'",periodtype,"\'));",sep="")
   result<-dfFromSQL(con,sql)
 return(result)
 }

#Main function which handles the plotting.
#con is the database connection
#IndicatorName is the name of the Indicator
#StartDate is the startdate
#baselayer is the baselayer
plotIndicator<-function(con,
IndicatorName,
StartDate,
periodtype="Yearly",
level=4,baselayer)
{
#First, get the desired indicator data
myDF<-dhisGetAggregatedIndicatorValues(con,
IndicatorName,StartDate,periodtype,level)
#Next, get the coordinates
coords<-dhisGetFacilityCoordinates(con,level)
#Merge the indicataors with the coordinates data frame
myDF<-merge(myDF,coords)
#We need to cast the new data fram to a spatial data
#frame in order to utilize plot
myDF<-SpatialPointsDataFrame(myDF[,
c("longitude","latitude")],myDF)
#Define some color scales
IndColors<-c("firebrick4","firebrick1","gold"
,"darkolivegreen1","darkgreen")
#Define the class breaks. In this case, we are going
#to use 6 quantiles
class<-classIntervals(myDF$value,n=6,style="quantile"
,pal=IndColors)
#Define a vector for the color codes to be used for the
#coloring of points by class
colCode<-findColours(class,IndColors)
#Go ahead and make the plot
myPlot<-plot.new()
#First, plot the base layer
plot(baselayer)
#Next, add the points data frame
points(myDF,col=colCode,pch=19)
#Add the indicator name to the title of the map
title(main=IndicatorName,sub=StartDate)
```

```
#Finally, return the plot from the function
return(myPlot) }
```

Up until this point, we have defined a few functions to help us make a map. We need to get the coordinates stored in the database and merge these with the indicator which we plan to map. We then retrieve the data from the aggregated indicator table, create a special type of data frame (SpatialPointsDataFrame), apply some styling to this, and then create the plot.

```
#Now we define the actual thing to do
#Lets get a connection to the database
con <- dbConnect(PostgreSQL(), user= "dhis", password="SomethingSecure",
 dbname="dhis")
#Define the name of the indicator to plot
MyIndicatorName<-"Total OPD Attendance"
MyPeriodType<-"Yearly"
#This should match the level where coordinates are stored
MyLevel<-4
#Given the startdate and period type, it is enough
#to determine the period
MyStartDate<-"2010-01-01"
#Get some Some Zambia district data from GADM
#This is going to be used as the background layer
con <- url("http://www.filefactory.com/file/c2a3898/n/ZMB_adm2_RData")
print(load(con))#saved as gadm object
#Make the map
plotIndicator(con,MyIndicatorName,MyStartDate,MyPeriodType,MyLevel,gadm)
```

The results of the plotIndicator function are shown below.

Total OPD Attendance

2010-01-01
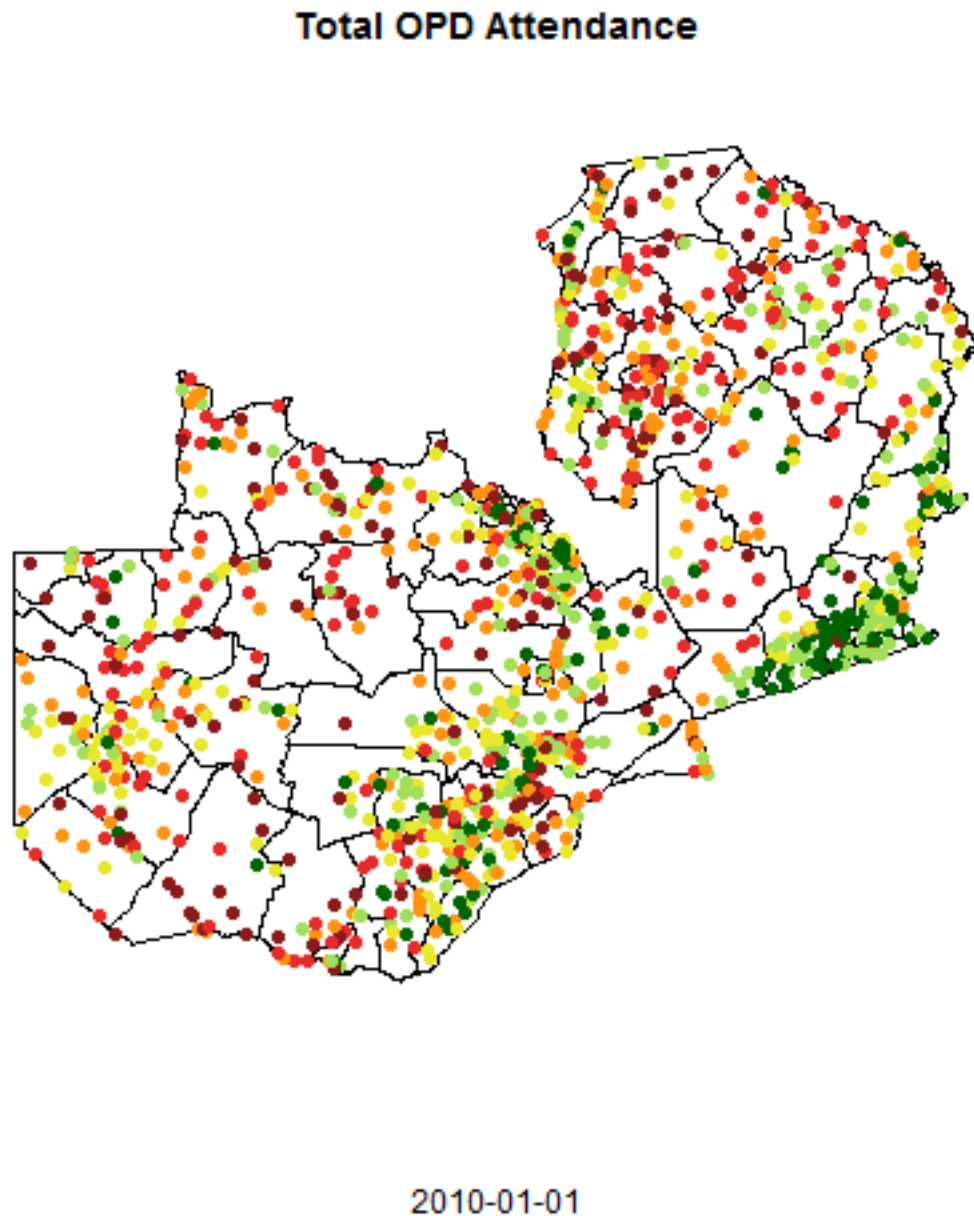
In this example, we showed how to use the RPostgreSQL library and other helper libraries(Maptools, ColorBrewer) to create a simple map from the DHIS2 data mart.

A.6. Using R, DHIS2 and the Google Visualization API

Google's Visualization API provides a very rich set of tools for the visualization of multi-dimensional data. In this simple example, we will show how to create a simple motion chart with the Google Visualization API using the "googleVis" R package. Full information on the package can be found here.. The basic principle, as with the other examples, is to get some data from the DHIS2 database, and bring it into R, perform some minor alterations on the data to make it easier to work with, and then create the chart. In this case, we will compare ANC1,2,3 data over time and see how they are related with a motion chart.

```
#Load some libraries
library(RPostgreSQL)
library(googleVis)
library(reshape)
#A small helper function to get a data frame from some SQL
```

```
dfFromSQL<-function (con,sql){
    rs<-dbSendQuery(con,sql)
    result<-fetch(rs,n=-1)
    return(result)
}

#Get a database connection
user<-"postgres"
password<-"postgres"
host<-"127.0.0.1"
port<-"5432"
dbname<-"dhis2_demo"
con <- dbConnect(PostgreSQL(), user= user,
password=password,host=host, port=port,dbname=dbname)
#Let's retrieve some ANC data from the demo database
sql<-"SELECT ou.shortname as province,
i.shortname as indicator,
extract(year from p.startdate) as year,
 a.value
FROM aggregatedindicatorvalue a
INNER JOIN  organisationunit ou on a.organisationunitid = ou.organisationunitid
INNER JOIN indicator i on a.indicatorid = i.indicatorid
INNER JOIN period p on a.periodid = p.periodid
WHERE a.indicatorid IN
(SELECT DISTINCT indicatorid from indicator where shortname ~*('ANC [123] Coverage'))
AND a.organisationunitid IN
 (SELECT DISTINCT idlevel2 from _orgunitstructure where idlevel2 is not null)
AND a.periodtypeid = (SELECT DISTINCT periodtypeid from periodtype where name =
 'Yearly')"
#Store this in a data frame
anc<-dfFromSQL(con,sql)
#Change these some columns to factors so that the reshape will work more easily

anc$province<-as.factor(anc$province)
anc$indicator<-as.factor(anc$indicator)
#We need the time variable as numeric
anc$year<-as.numeric(as.character(anc$year))
#Need to cast the table into a slightly different format
anc<-cast(anc,province + year ~ indicator)
#Now, create the motion chart and plot it
M<-gvisMotionChart(anc,idvar="province",timevar="year")
plot(M)
```

The resulting graph is displayed below.
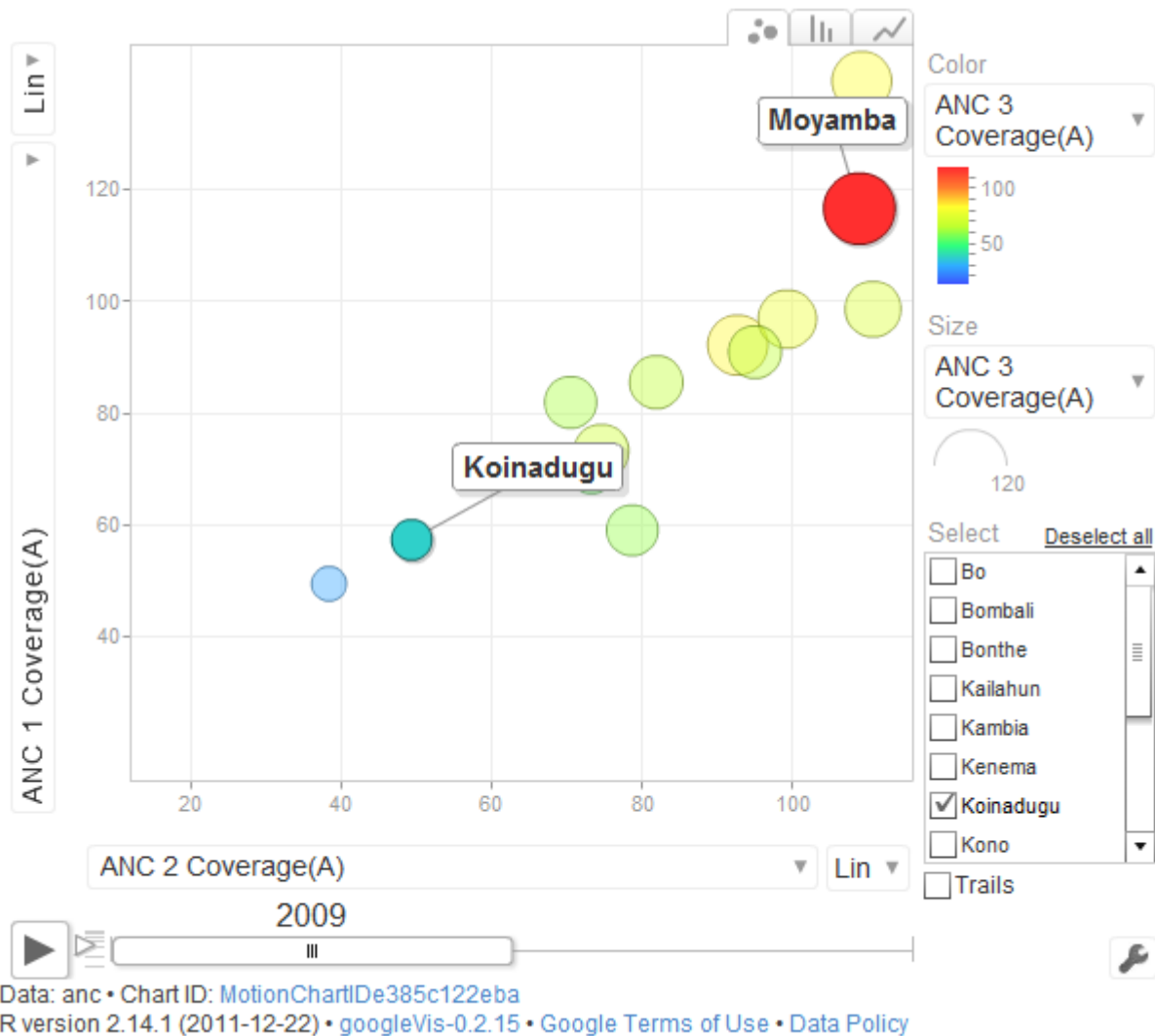
Using packages like brew or Rapache, these types of graphs could be easily integrated into external web sites. A fully functional version of the chart shown above can be accessed here.

A.7. Using PL/R with DHIS2

The procedural language for R is an extension to the core of PostgreSQL which allows data to be passed from the database to R, where calculations in R can be performed. The data can then be passed back to the database for further processing.. In this example, we will create a function to calculate some summary statistics which do not exist by default in SQL by using R. We will then create an SQL View in DHIS2 to display the results. The advantage of utilizing R in this context is that we do not need to write any significant amount of code to return these summary statistics, but simply utilize the built-in functions of R to do the work for us.

First, you will need to install PL/R, which is described in detail here.. Following the example from the PL/R site, we will create some custom aggregate functions as detailed here. We will create two functions, to return the median and the skewness of a range of values.

```
CREATE OR REPLACE FUNCTION r_median(_float8) returns float as '
  median(arg1)
' language 'plr';

CREATE AGGREGATE median (
  sfunc = plr_array_accum,
  basetype = float8,
  stype = _float8,
```

```
   finalfunc = r_median
);

CREATE OR REPLACE FUNCTION r_skewness(_float8) returns float as '
   require(e1071)
   skewness(arg1)
' language 'plr';

CREATE AGGREGATE skewness (
   sfunc = plr_array_accum,
   basetype = float8,
   stype = _float8,
   finalfunc = r_skewness
);
```

Next, we will define an SQL query which will be used to retrieve the two new aggregate functions (median and skewness) which will be calculated using R. In this case, we will just get a single indicator from the data mart at the district level and calculate the summary values based on the name of the district which the values belong to. This query is very specific, but could be easily adapted to your own database.

```
SELECT  ou.shortname,avg(dv.value),
median(dv.value),skewness(dv.value) FROM aggregatedindicatorvalue dv
INNER JOIN period p on p.periodid = dv.periodid
INNER JOIN organisationunit ou on
dv.organisationunitid = ou.organisationunitid
WHERE dv.indicatorid = 112670
AND dv.level = 3
AND dv.periodtypeid = 3
AND p.startdate >='2009-01-01'
GROUP BY ou.shortname;
```

We can then save this query in the form of SQL View in DHIS2. A clipped version of the results are shown below.

shortname	avg	median	skewness
Mambwe District	4.003571428571428	2.7	1.06755380878575
Serenje District	1.0892857142857142	0.6	1.1161464397199
Siavonga District	0.44642857142857145	0.35	0.781235716148461
Nakonde District	0.15000000000000005	0.1	2.70015899330264
Milenge District	3.410714285714286	3.05	0.288213413218391
Ndola District	1.2857142857142854	1.35	0.803609762712828
Masaiti District	1.467857142857143	0.65	1.41409131466381
Senanga District	0.04642857142857143	0.0	1.61843107173922

In this simple example, we have shown how to use PL/R with the DHIS2 database and web interface to display some summary statistics using R to perform the calculations.

A.8. Using this DHIS2 Web API with R

DHIS2 has a powerful Web API which can be used to integrate applications together. In this section, we will illustrate a few trivial examples of the use of the Web API, and how we can retrieve data and metadata for use in R. The Web API uses basic HTTP authentication (as described in the Web API section of this document). Using two R packages "RCurl" and "XML", we will be able to work with the output of the API in R. In the first example, we will get some metadata from the database.

```
#We are going to need these two libraries
require(RCurl)
require(XML)
```

```
#This is a URL endpoint for a report table which we can
#get from the WebAPI.

url<-"https://apps.dhis2.org/dev/api/reportTables/KJFbpIymTAo/data.csv"
#Lets get the response and we do not need the headers
#This site has some issues with its SSL certificate
#so lets not verify it.
response<-getURL(url,userpwd="admin:district"
,httpauth = 1L, header=FALSE,ssl.verifypeer = FALSE)
#Unquote the data
data<-noquote(response)
#here is the data.
mydata<-read.table(textConnection(data),sep=",",header=T)
head(mydata)
```

Here, we have shown how to get some aggregate data from the DHIS2 demo database using the DHIS2's Web API.

In the next code example, we will retrieve some metadata, namely a list of data elements and their unique identifiers.

```
#Get the list of data elements. Turn off paging and links
#This site has some issues with its SSL certificate
#so lets not verify it.
url<-"https://apps.dhis2.org/dev/api/dataElements.xml?
paging=false&links=false"
response<-getURL(url,userpwd="admin:district",
httpauth = 1L, header=FALSE,ssl.verifypeer = FALSE)
#We ned to parse the result
bri<-xmlParse(response)
#And get the root
r<-xmlRoot(bri)
#Parse out what we need explicitly, in this case from the first node
#Just get the names and ids as separate arrays
de_names<-xmlSApply(r[['dataElements']],xmlGetAttr,"name")
de_id<-xmlSApply(r[['dataElements']],xmlGetAttr,"id")
#Lets bind them together
#but we need to be careful for missing attribute values
foo<-cbind(de_names,de_id)
#Recast this as a data frame
data_elements<-as.data.frame(foo,
stringsAsFactors=FALSE,row.names=1:nrow(foo))
head(data_elements)
```

Note that the values which we are interested in are stored as XML attributes and were parsed into two separate matrices and then combined together into a single data frame.