

DHIS2 Developer Manual



© 2006-2015
DHIS2 Documentation Team

Revision 1699
Version 2.21 2016-03-01 10:54:23

Warranty: THIS DOCUMENT IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS MANUAL AND PRODUCTS MENTIONED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

License: Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the source of this documentation, and is available here online: <http://www.gnu.org/licenses/fdl.html>.

1. Web API	1
1.1. Introduction	1
1.2. Authentication	1
1.2.1. Basic Authentication	1
1.2.2. OAuth2	2
1.2.2.1. Adding a client using the web-api	2
1.2.2.2. Grant type password	2
1.2.2.3. Grant type refresh_token	3
1.2.2.4. Grant type authorized_code	3
1.3. Error and info messages	3
1.4. Date and period format	4
1.5. Browsing the Web API	5
1.5.1. Translation	5
1.6. Working with the metadata API	5
1.6.1. Content types	5
1.6.2. Query parameters	6
1.6.3. Available strategies for import	8
1.6.4. Examples	8
1.7. Metadata object filter	9
1.8. Metadata field filter	10
1.8.1. Field transformers	11
1.8.2. Field converters	11
1.9. Metadata create, read, update, delete, validate	11
1.9.1. Create / update parameters	12
1.9.2. Creating and updating objects	12
1.9.3. Deleting objects	13
1.9.4. Adding and removing objects to/from collections	13
1.9.5. Validating payloads	14
1.9.6. Partial updates	14
1.10. CSV metadata import	15
1.10.1. Data elements	15
1.10.2. Organisation units	16
1.10.3. Validation rules	17
1.10.4. Option sets	18
1.10.5. Other objects	18
1.11. File resources	19
1.11.1. File resource constraints	19
1.12. Data values	20
1.12.1. Sending data values	20
1.12.2. Sending bulks of data values	22
1.12.2.1. Identifier schemes	23
1.12.3. CSV data value format	24
1.12.4. Generating data value set template	24
1.12.5. Sending, reading and deleting individual data values	25
1.12.5.1. Working with file data values	26
1.12.6. Reading data values	26
1.13. ADX formatted data	28
1.13.1. The adx root element	29
1.13.2. The group element	29
1.13.3. Data values	29
1.13.4. POSTing data	30
1.14. Events	30
1.14.1. Sending events	30
1.14.2. CSV Import / Export	32
1.14.3. Querying and reading events	33
1.14.3.1. Examples	35
1.15. Forms	35

1.16. Validation	36
1.17. Data integrity	37
1.17.1. Running data integrity	37
1.17.2. Fetching the result	37
1.18. Indicators	37
1.18.1. Aggregate indicators	37
1.18.2. Program indicators	38
1.18.3. Expressions	38
1.19. Complete data set registrations	39
1.19.1. Completing and un-completing data sets	39
1.19.2. Reading complete data set registrations	39
1.20. Data approval	40
1.21. Messages	42
1.21.1. Writing and reading messages	42
1.21.2. Managing messages	44
1.22. Interpretations	45
1.22.1. Reading interpretations	45
1.22.2. Writing interpretations	46
1.22.3. Creating, updating and removing interpretation comments	46
1.23. Viewing analytical resource representations	47
1.24. Plugins	48
1.24.1. Embedding pivot tables with the Pivot Table plug-in	48
1.24.2. Embedding charts with the Visualizer chart plug-in	51
1.24.3. Embedding maps with the GIS map plug-in	55
1.24.4. Creating a chart carousel with the carousel plug-in	59
1.25. SQL views	60
1.25.1. Criteria	61
1.25.2. Variables	61
1.26. Dashboard	61
1.26.1. Browsing dashboards	61
1.26.2. Searching dashboards	62
1.26.3. Creating, updating and removing dashboards	63
1.26.4. Adding, moving and removing dashboard items and content	63
1.27. Analytics	64
1.27.1. Dimensions and items	64
1.27.2. Request query parameters	66
1.27.3. Response formats	68
1.27.4. Constraints	70
1.27.5. Debugging	70
1.28. Event analytics	71
1.28.1. Dimensions and items	71
1.28.2. Request query parameters	71
1.28.3. Event query analytics	73
1.28.3.1. Filtering	73
1.28.3.2. Response formats	74
1.28.4. Event aggregate analytics	76
1.28.4.1. Ranges / legend sets	77
1.28.4.2. Response formats	77
1.29. Geo features	79
1.29.1. GeoJSON	79
1.30. Generating resource and analytics tables	79
1.31. Maintenance	80
1.32. System resource	81
1.32.1. Generate identifiers	81
1.32.2. View system information	81
1.32.3. Check if username and password combination is correct	82
1.32.4. View asynchronous task status	82

1.32.5. Get appearance information	83
1.33. Users	83
1.33.1. User query	83
1.33.2. User account create and update	84
1.33.3. User account invitations	84
1.33.4. User replication	86
1.34. Current user information and associations	86
1.35. System settings	87
1.36. User settings	87
1.37. Organisation units	88
1.38. Static content	88
1.39. Configuration	89
1.40. Internationalization	90
1.41. SVG conversion	90
1.42. Tracked entity management	90
1.43. Tracked entity instance management	91
1.43.1. Creating a new tracked entity instance	91
1.43.2. Updating a tracked entity instance	91
1.43.3. Deleting a tracked entity instance	92
1.43.4. Enrolling a tracked entity instance into a program	92
1.43.5. Update strategies	92
1.44. Enrollment instance query	92
1.44.1. Request syntax	93
1.44.2. Response format	94
1.45. Tracked entity instance query	94
1.45.1. Request syntax	95
1.45.2. Response format	97
1.46. Tracked entity instance grid query	99
1.46.1. Request syntax	99
1.46.2. Response format	102
1.47. Email	103
1.47.1. System notification	103
1.47.2. Test message	104
1.48. Sharing	104
1.49. Scheduling	105
1.50. Schema Resource	105
1.51. UI customization with Javascript and CSS files	105
1.52. Synchronization	106
1.52.1. Data push	106
1.52.2. Metadata pull	106
1.53. FRED API	106
1.54. Data store	107
1.54.1. Get keys and namespaces	107
1.54.2. Create and update values	108
1.54.3. Delete keys	108
1.55. Metadata repository	109
2. Apps	111
2.1. Purpose of packaged Apps	111
2.2. Creating Apps	111
2.3. Configuring DHIS 2 for Apps Installation	112
2.4. Installing Apps into DHIS 2	112
2.5. Launching Apps	113
2.6. Web-API for Apps	113
2.7. Adding the DHIS 2 menu to your app	114
3. Infrastructure	117
3.1. Release process	117
DHIS 2 Technical Architecture	119

1. Overview	119
2. Technical Requirements	119
3. Project Structure	119
4. Project Dependencies	120
5. The Data Model	121
6. The Persistence Layer	122
7. The Business Layer	123
7.1. The JDBC Service Project	123
7.2. The Data Mart Project	125
7.3. The Reporting Project	127
7.3.1. Report table	127
7.3.2. Chart	128
7.3.3. Data set completeness	128
7.3.4. Document	129
7.3.5. Pivot table	130
7.3.6. The External Project	130
7.4. The System Support Project	130
7.4.1. DeletionManager	130
8. The Presentation Layer	131
8.1. The Portal	131
8.1.1. Module Assembly	131
8.1.2. Portal Module Requirements	131
8.1.3. Common Look-And-Feel	131
8.1.4. Main Menu	132
9. Framework Stack	132
9.1. Application Frameworks	132
9.2. Development Frameworks	132
10. Definitions	132
A. R and DHIS 2 Integration	133
A.1. Introduction	133
A.2. Using ODBC to retrieve data from DHIS2 into R	133
A.3. Using R with MyDatamart	135
A.4. Mapping with R and PostgreSQL	137
A.5. Using R, DHIS2 and the Google Visualization API	140
A.6. Using PL/R with DHIS2	142
A.7. Using this DHIS2 Web API with R	143

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 DHIS 2. 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 DHIS 2 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 DHIS 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. 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 DHIS 2 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/me
```

1.2.1. Basic Authentication

The DHIS 2 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

DHIS 2 support the OAuth2 protocol, which is an open standard for authorization, it allows 3rd party clients to connect on behalf of a DHIS 2 user, and get a re-usable bearer token for multiple requests to the web-api (without requiring getting the users permission again). We do not support fine-grained OAuth2 roles, rather we gives the application access based on the users DHIS 2 user role.

Each client for which you want to allow OAuth 2 authentication must be registered in DHIS 2. To add a new OAuth2 client, go to *settings => OAuth2 Clients*, click add new and at least the name of client, and the grant types you want supported.

1.2.2.1. Adding a client using the web-api

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

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

```
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 authentication 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/
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 authorized_code

Authorized code grant type is the recommended approach if you don't want to store the user credentials externally. It allows DHIS 2 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_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=authorized_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.

Name	Description
statusCode	HTTP Status code for this response, see RFC 2616 (Section 10) for more information.
status	DHIS 2 status, possible values are <i>OK</i> <i>WARNING</i> <i>ERROR</i> , where OK 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 technical developer friendly message (not currently in use).
response	Extension point for future extension to the WebMessage format. This will be documented when it starts being used.

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
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,
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. 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 1.3. Query parameters

Param	Option values	Default option	Description
links	true false	true	Indicates whether to include links to relevant elements.
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	propertyName:asc/ desc		Order the output using a specified order, only properties that are both persisted and simple (no collections, idObjects etc) are supported.

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

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

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

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

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

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

1.5.1. Translation

Support for I18n translation in the web-api was added in 2.19 release. It is supported by two parameters:

Table 1.4. Translate options

Parameter	Values	Description
translate	true/false	Translate web-api output, display* properties will be used (displayName, displayShortName, displayDescription)
locale	Locale to use	Translate web-api using a specified locale (implies translate=true)

1.6. 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.

1.6.1. Content types

The Web API offers several content types for metadata.

Table 1.5. 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

1.6.2. Query parameters

The following query parameters are available for customizing your request.

Table 1.6. 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 depends on assumeTrue)	See <i>/api</i> for available resources. 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

Param	Type	Required	Options (default first)	Description
				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, and not touched on update).
mergeStrategy	enum	false	REPLACE, MERGE	Strategy for merging of objects when doing updates. REPLACE will just overwrite the property with the new value provided, MERGE will only set the property if its not null (only if the property was provided).
async	boolean	false	false true	Indicates whether 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

Param	Type	Required	Options (default first)	Description
				back to you). For large imports, the client might time out, so <code>async=true</code> is recommended, and the client connection will be dropped when the payload is uploaded.

1.6.3. Available strategies for import

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

Table 1.7. 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.

1.6.4. 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 asynchronous 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.

1.7. 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 1.8. Available Operators

Operator	Types	Value required	Description	
eq	string boolean integer float enum collection (checks for size) date	true	Equality	
ne	string boolean integer float collection (checks for size) date	true	Inequality	
like / ilike	string	true	Case insensitive string matching	
nlike	string	true	Case insensitive string not matching	
startsWith	string	true	Case insensitive string matching	
endsWith	string	true	Case insensitive string matching	
gt	string boolean integer float collection (checks for size) date	true	Greater than	
ge	string boolean integer float collection (checks for size) date	true	Greater than or equal	
lt	string boolean integer float collection (checks for size) date	true	Less than	
le	string boolean integer float collection (checks for size) date	true	Less than or equal	
null	all	false	Property is null	
empty	collection	false	Collection is empty	
in	string boolean integer float date	true	Find objects 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/dataElements?filter=id:eq:ID1&filter=id:eq:ID2
```

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

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

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

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

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

```
/api/dataElements.json?filter=dataElementGroups.id:eq:qfxEYY9xA16
```

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

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

1.8. 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/indicators?fields=id,name
```

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

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

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

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

Table 1.9. 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 include fields.
, <object>[]	Include all fields on a certain object, if applied to a collection, it will include all fields on all objects on that collection.
:<preset>	Alias to select multiple fields. Three presets are currently available, see table below for descriptions.

Table 1.10. 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/dataSets?fields=:all,!organisationUnits
```

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

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

Example: Include nameable properties from all indicators:

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

1.8.1. Field transformers

In DHIS 2.17 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/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.*

1.8.2. Field converters

In DHIS 2.17 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 1.11. 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

1.9. 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).

1.9.1. Create / update parameters

The following query parameters are available for customizing your request.

Table 1.12. Available Query Filters

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

1.9.2. 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/constants" -X POST -H "Content-Type:
application/json" -u user:password
```

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

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

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

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

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

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

1.9.3. 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/constants/ID
```

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

1.9.4. 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/{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/dataElementGroups/IDA/dataElements/IDB
```

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

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

1.9.5. 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."
  }
]
```

1.9.6. 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/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/dataElements/fbfJHSPpUQD
```

1.10. CSV metadata import

DHIS 2 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
- Option sets

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

1.10.1. Data elements

Table 1.13. Data Element CSV Format

Column	Required	Value (default first)	Description
Name	Yes		Name. Max 230 char. 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	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.
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	Operator indicating how to aggregate data in the time dimension. Max 16 char.

Column	Required	Value (default first)	Description
		VARIANCE MIN MAX NONE	
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,formname,domain, type, numbertype, texttype, aggregationoperator
"Women participated in skill development training",, "D0001", "Women participated
development training"
"Women participated in community organizations",, "D0002", "Women participated community
organizations"
```

1.10.2. Organisation units

Table 1.14. 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.
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.

Column	Required	Value (default first)	Description
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"
```

1.10.3. Validation rules

Table 1.15. 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	
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.

Column	Required	Value (default first)	Description
Right side null if blank	No	false true	Boolean.

1.10.4. Option sets

Table 1.16. 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",,"Unknown",,"UNKNOWN"
"Result",,"High",,"HIGH"
"Result",,"Medium",,"MEDIUM"
"Result",,"Low",,"LOW"
"Impact","cJ82jd8sd32","IMPACT","Great",,"GREAT"
"Impact","cJ82jd8sd32","IMPACT","Medium",,"MEDIUM"
"Impact","cJ82jd8sd32","IMPACT","Poor",,"POOR"
```

1.10.5. Other objects

Table 1.17. 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.

An example for category options looks like this:

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


1.11. 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*).

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/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 1.12.5.1, “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/fileResources/xm4JwRwke0i -H "Accept: application/json"
```

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

1.11.1. 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.

1.12. Data values

This section is about sending and reading data values.

1.12.1. 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 DHIS 2 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 DHIS 2 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/dataValueSets>. The format looks like this:

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

JSON is supported in this format:

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

CSV is supported in this format:

```
"dataelement","period","orgunit","catoptcombo","attroptcombo","value","storedby","lastupd","comment"
"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> and follow the embedded link pointing at the *dataSets* resource located at <http://play.dhis2.org/demo/api/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/dataSets/pBOMPrpglQX>. 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/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="pBOMPrpglQX"
  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": "pBOMPrpglQX",
  "completeData": "2014-02-03",
  "period": "201401",
  "orgUnit": "DiszpKrYNg8",
  "dataValues": [
    { "dataElement": "f7n9E0hX8qk", "value": "12" },
    { "dataElement": "Ix2HsbDMLea", "value": "14" },
    { "dataElement": "eY5ehpbEsB7", "value": "16" }
  ]
}
```

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/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/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 DHIS 2 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.

1.12.2. 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/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/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/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.

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

Table 1.18. Import parameters

Parameter	Values (default first)	Description
dataElementIdScheme	id name code	Property of the data element object to use to map the data values.
orgUnitIdScheme	id name code	Property of the org unit object to use to map the data values.
idScheme	id name code	Property of all objects including data elements, org units and category option combos, to use to map the data values.
dryRun	false true	Whether to save changes on the server or just return the import summary.
preheatCache	false true	Whether to preheat data element and organisation unit caches with all objects.
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.

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

```
https://play.dhis2.org/demo/api/dataValueSets?
dataElementIdScheme=code&orgUnitIdScheme=name&dryRun=true&importStrategy=new
```

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="new">
  ..
</dataValueSet>
```

1.12.2.1. Identifier schemes

Regarding the id schemes, by default the identifiers used in the XML messages uses the DHIS 2 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 identifier property of these metadata objects. Identifier schemes can be specified in the XML message as well as in the request as query parameters. To specify it in the XML payload you can do this:

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

```
</dataValueSet>
```

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

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

1.12.3. 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 1.19. CSV format of DHIS 2

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 DHIS 2 is seen below.

```
"dataelement","period","orgunit","categoryoptioncombo","attroptioncombo","value","storedby","time
"DUSpd8Jq3M7","201202","gP6hn503KUX","Pr1t0C1RF0s",,"7","bombali","2010-04-17",,"false"
"DUSpd8Jq3M7","201202","gP6hn503KUX","V6L425pT3A0",,"10","bombali","2010-04-17",,"false"
"DUSpd8Jq3M7","201202","OjTS752GbZE","V6L425pT3A0",,"9","bombali","2010-04-06",,"false"
```

1.12.4. Generating data value set template

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

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

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

Table 1.20. Data values query parameters

Query parameter	Required	Description
period	No	Period to use, will be included without any checks.

Query parameter	Required	Description
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.

1.12.5. 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/dataValues
```

The following query parameters are supported for this resource:

Table 1.21. 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/dataValues?
de=s46m5MS0hXu&pe=201301&ou=DiszpKrYNg8&co=Pr1t0C1RF0s&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/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/dataValues?
de=s46m5MS0hXu&pe=201301&ou=DiszpKrYNg8&co=Pr1t0C1RF0s"
-X GET -u admin:district -v
```

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

1.12.5.1. 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/fileResources* endpoint as described in [Section 1.11, “File resources”](#).
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 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.

1.12.6. 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 1.22. Data value set query parameters

Parameter	Description
dataSet	Data set 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 after the given time stamp.
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)

Assuming that we have posted data values to DHIS 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/dataValueSets?
dataSet=pBOMPrpglQX&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 precedence over the start and end date parameters. An example looks like this:

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

The response will look like this:

```
<?xml version='1.0' encoding='UTF-8'?>
<dataValueSet xmlns="http://dhis2.org/schema/dxf/2.0" dataSet="pBOMPrpglQX"
  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/dataValueSets.json?
dataSet=pBOMPrpglQX&period=201401&orgUnit=DiszpKrYNg8
```

The response will look something like this:

```
{
  "dataSet": "pBOMPrpglQX",
  "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/dataValueSets.csv?
dataSet=pBOMPrpglQX&period=201401&orgUnit=DiszpKrYNg8
```

The response will look like this:

```
dataelement,period,orgunit,categoryoptioncombo,attributeoptioncombo,value,storedby,lastupdated,comment
f7n9E0hX8qk,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,12,system,2015-04-05T19:58:12.000,comment:
Ix2HsbDMLea,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,14,system,2015-04-05T19:58:12.000,comment:
eY5ehpbEsB7,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,16,system,2015-04-05T19:58:12.000,comment:
FTTrcoaog83,201401,DiszpKrYNg8,bRowv6yZOF2,bRowv6yZOF2,12,system,2014-03-02T21:45:05.519,comment:
```

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.

1.13. ADX formatted data

From version 2.20 we have included support for an upcoming international standard for aggregate data exchange called ADX. ADX is developed and maintained by the Quality Research and Public Health committee of the IHE (Integrating the HealthCare Enterprise). The wiki page detailing QRPH activity can be found at wiki.ihe.net. ADX is still under active development and is due to be published before the end of 2015. Nevertheless, the data format for aggregate data values is currently fairly stable and is unlikely to change much from what is described here. Note that what is implemented currently in DHIS 2 is the functionality to read adx formatted data, i.e. what is described as a Content Consumer actor 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>
```

1.13.1. The adx root element

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

1.13.2. The group element

Unlike dxf2, adx requires that the data values 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 (An adx message can also be coded using uids using the same idScheme mechanism used with dxf2. More on this when we describe posting the data below).

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

1.13.3. 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 characters other than '_' and it may not start with a letter. The example above shows examples of 'good' category codes ('GENDER' and 'HIV_AGE').

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

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

- The system producing the data does not have to be 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.

1.13.4. POSTing data

DHIS2 exposes an endpoint for POST adx data at */ohie/dataValueSets*. 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"
  -d @data.xml "https://play.dhis2.org/demo/ohie/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.

1.14. Events

This section is about sending and reading events.

1.14.1. Sending events

DHIS 2 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 DHIS 2 you must interact with the *events* resource. The approach to sending events is similar to sending aggregate data values. You will need a *program* which can be looked up using the *programs* resource, an *orgUnit* which can be looked up using the *organisationUnits* resource, and a list of valid data element identifiers which can be looked up using the *dataElements* resource. For events with registration, a *tracked entity instance* identifier is required, read about how to get this in the section about the *trackedEntityInstances* resource. For sending events to programs with multiple stages, you will need to also include the *programStage* identifier, the identifiers for programStages can be found in the *programStages* resource.

A simple single event with no registration example payload in XML format where we send events from the "Inpatient morbidity and mortality" program for the "Ngelehun CHC" facility in the demo database can be seen below:

```
<?xml version="1.0" encoding="utf-8"?>
<event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8" eventDate="2013-05-17"
  status="COMPLETED" storedBy="admin">
  <coordinate latitude="59.8" longitude="10.9" />
  <dataValues>
    <dataValue dataElement="qrur9Dvnyt5" value="22" />
    <dataValue dataElement="oZg33kd9taw" value="Male" />
    <dataValue dataElement="msodh3rEMJa" value="2013-05-18" />
  </dataValues>
</event>
```

To perform some testing we can save the XML payload as a file called *event.xml* and send it as a POST request to the events resource in the API using curl with the following command:

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

The same payload in JSON format looks like this:

```
{
  "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/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 version="1.0" encoding="utf-8"?>
<events>
  <event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8" eventDate="2013-05-17"
status="COMPLETED" storedBy="admin">
    <coordinate latitude="59.8" longitude="10.9" />
    <dataValues>
      <dataValue dataElement="qrur9Dvnyt5" value="22" />
      <dataValue dataElement="oZg33kd9taw" value="Male" />
    </dataValues>
  </event>
  <event program="eBAyeGv0exc" orgUnit="DiszpKrYNg8" eventDate="2013-05-17"
status="COMPLETED" storedBy="admin">
    <coordinate latitude="59.8" longitude="10.9" />
    <dataValues>
      <dataValue dataElement="qrur9Dvnyt5" value="26" />
      <dataValue dataElement="oZg33kd9taw" value="Female" />
    </dataValues>
  </event>
</events>

```

You will receive an import summary with the response which can be inspected in order to get information about the outcome of the request, like how many values were imported successfully. The payload in JSON format looks like this:

```

{
  "events": [
    {
      "program": "eBAyeGv0exc",
      "orgUnit": "DiszpKrYNg8",
      "eventDate": "2013-05-17",
      "status": "COMPLETED",
      "storedBy": "admin",
      "coordinate": {
        "latitude": "59.8",
        "longitude": "10.9"
      },
      "dataValues": [
        { "dataElement": "qrur9Dvnyt5", "value": "22" },
        { "dataElement": "oZg33kd9taw", "value": "Male" }
      ]
    },
    {
      "program": "eBAyeGv0exc",
      "orgUnit": "DiszpKrYNg8",
      "eventDate": "2013-05-17",
      "status": "COMPLETED",
      "storedBy": "admin",
      "coordinate": {
        "latitude": "59.8",
        "longitude": "10.9"
      }
    }
  ]
}

```

```

    },
    "dataValues": [
      { "dataElement": "qrur9Dvnyt5", "value": "26" },
      { "dataElement": "oZg33kd9taw", "value": "Female" }
    ]
  }
}

```

(From 2.13) 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.

OrgUnit matching: By default the orgUnit parameter will match on the ID (of the orgUnit, but from 2.15 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* (ID and UID will both matchUIDs).

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/events/ID" -H "Content-Type:application/xml" -u admin:district -v
```

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

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/events/ID" -u admin:district -v
```

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

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

The table below describes the meaning of each element. Most elements should be fairly self-explanatory.

Table 1.23. Events resource format

Parameter	Type	Required	Options (default first)
programId	string	true	
organisationUnitId	string	true	
eventDate	date	true	
status	enum	false	ACTIVE COMPLETED VISITED SCHEDULE
storedBy	string	false	Defaults to current user
coordinate	double	false	
dataElementId	string	true	
value	string	true	

1.14.2. CSV Import / Export

In addition to XML and JSON for event import/export, in DHIS 2.17 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 1.24. 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
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

1.14.3. 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.

Table 1.25. 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, can be ACTIVE COMPLETED CANCELLED
followUp	boolean	false	Whether event is considered for follow up in program, can be true false or omitted.

Key	Type	Required	Description
trackedEntityInstance	identifier	false	Identifier of tracked entity instance
orgUnit	identifier	true	Identifier of organisation unit
ouMode	enum	false	Org unit selection mode, can be SELECTED CHILDREN DESCENDANTS
startDate	date	false	Only events newer than this date
endDate	date	false	Only events older than this date
status	enum	false	Status of event, can be ACTIVE COMPLETED VISITED SCHEDULED OVERDUE SKIPPED
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	false	Number of items in each page
totalPages	boolean	false	Indicates whether to include the total number of pages in the paging response.
skipPaging	boolean	false	Indicates whether to skip paging in the query and return all events.
dataElementIdScheme	string	false	Data element ID scheme to use for export, valid options are UID 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

Key	Type	Required	Description
programStageIdScheme	string	false	Program Stage ID scheme to use for 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.

1.14.3.1. Examples

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

```
api/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/events.json?orgUnit=06uvpzGd5pu&ouMode=DESCENDANTS
```

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

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

Query for all events with a certain program and organisation unit for a specific tracked entity instance:

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

1.15. 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 1.26. Form query parameters

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

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

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

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

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

To retrieve the form including metadata in JSON:

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

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

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

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

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

1.16. 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/validation/dataSet/QX4ZTUbt3a.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 the *validationRules* resource like this:

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

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

Table 1.27. 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 1.28. Missing value strategies

Value	Description
SKIP_IF_ANY_VALUE_MISSING	Skip validation rule if any data value is missing

Value	Description
SKIP_IF_ALL_VALUES_MISSING	Never skip validation rule if all data values are missing
NEVER_SKIP	Never skip validation rule irrespective of missing data values

1.17. 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.

1.17.1. 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/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.

1.17.2. 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/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.

1.18. Indicators

This section describes indicators and indicator expressions.

1.18.1. Aggregate indicators

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

```
api/indicators
```

Indicators represent expressions which can be calculated and presented as a result. The indicator expressions are split into a numerator and denominator. The numerators and denominators are mathematical expressions which can contain references to data elements, 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 1.29. Indicator variables

Variable	Description
#{<dataelement-id>.<catoptcombo-id>}	Refers to a combination of a data element and a category option combination.

Variable	Description
#{<dataelement-id>}	Refers to the total value of a data element across all category option combinations.
C{<constant-id>}	Refers to a constant.
OUG{<orgunitgroup-id>}	Refers to the count of organisation units in 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
```

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

```
( 2 * #{P3jJH5Tu5VC.S34ULMchMca} ) / ( #{FQ2o8UBlcrS.S34ULMchMca} - 200 ) * 25
```

1.18.2. Program indicators

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

```
api/programIndicators
```

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

Table 1.30. 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{<variable-id>}	Refers to a program variable.
C{<constant-id>}	Refers to a constant.

The syntax looks like this:

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

A corresponding example looks like this:

```
#{A03MvHHogjR.a3kGcGDCuk6} + A{OvY4VVhSDeJ} + V{incident_date} + C{bCqvfpR02Im}
```

1.18.3. 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/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"
}
```

1.19. 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.

1.19.1. 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/completeDataSetRegistrations
```

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

Table 1.31. 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 cc)	Attribute option identifiers, separated with ; for multiple values (for locking check)
multiOu	No (default false)	Whether registration applies to sub units

1.19.2. 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 1.32. 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

Parameter	Description
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/completeDataSetRegistrations?
dataSet=pBOMPrpglQX&dataSet=pBOMPrpglQX&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*.

1.20. 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.

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

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

Table 1.33. Data approval query parameters

Query parameter	Required	Description
ds	Yes	Data set identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
cog	No	Attribute category option group identifier
cp	No	Attribute category option identifier(s), repeat the parameter for multiple values

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 1.34. 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 1.35. 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: <ul style="list-style-type: none"> • Data is approved at a higher level. • Data is approved for wider scope of category options. • Data is approved for all sub-periods in selected period. In the first two cases, there is a single data approval object that covers the selection. In the third case there is not.
ACCEPTED_HERE	Data is approved and accepted here (so could be unapproved here.)
ACCEPTED_ELSEWHERE	Data is approved and accepted, but elsewhere.

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

- The organisation unit might not be at an approval level. The approval status is determined by whether data is approved at an approval level for an ancestor of the organisation unit.
- You may specify individual attribute category options. The approval status is determined by whether data is approved for an attribute category option combination that includes one or more of these options.
- You may specify a time period that is longer than the period for the data set at which the data is entered and approved. The approval status is determined by whether the data is approved for all the data set periods within the period you specify.

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.

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.

These requests contain the following parameters:

Table 1.36. Data approval action parameters

Action parameter	Required	Description
ds	Yes	Data set identifier
pe	Yes	Period identifier
ou	Yes	Organisation unit identifier
cog	No	Attribute category option group identifier. Required if approving for an approval level that contains a category option group set, otherwise must not be present.

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 category option group (if specified) must be a member of an approval level's category option group set (if specified) for an approval level with the same organisation unit 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.

1.21. Messages

DHIS 2 features a mechanism for sending messages for purposes such as user feedback, notifications and general information to users. Messages are delivered to the DHIS 2 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 *DHIS 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.

1.21.1. 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/users>. We continue by following link to the mobile user at <http://play.dhis2.org/demo/api/users/PhzytPW3g2J> where we learn that her

identifier is *PhzytPW3g2J*. We are now ready to put our XML message together to form a message where we want to ask the mobile user whether she has reported data for January 2014:

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

To test this we save the XML content into a file called *message.xml*. We use cURL to dispatch the message to the DHIS 2 demo instance where we indicate that the content-type is XML and authenticate as the *admin* user:

```
curl -d @message.xml "https://play.dhis2.org/demo/api/messageConversations" -H
  "Content-Type:application/xml" -u admin:district -X POST -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/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 the message which was just sent by dispatching a GET request to the *messageConversations* resource. We supply an *Accept* header with *application/xml* as the value to indicate that we are interested in the XML resource representation and we authenticate as the *mobile* user:

```
curl "https://play.dhis2.org/demo/api/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="DHIS version 2.7 is deployed" id="GDBqVfkmp2"
    link="https://play.dhis2.org/demo/api/messageConversations/GDBqVfkmp2"/>
```

```
</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/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/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.

1.21.2. 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/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/messageConversations?mc=WzMRrCosqc0&mc=lxCjiigqrJm" -X DELETE -u admin:district -v
```

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

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

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

Messages carry a boolean *read* property. This allows tracking whether a user has seen (opened) a message or not. In a typical application scenario (e.g. the DHIS 2 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" ] }
```

1.22. Interpretations

For certain analysis-related resources in DHIS, like charts, maps and report tables, one can write and share a data interpretation. An interpretation is simply a link to the relevant resource together with a text expressing some insight about the data. Interpretations access control follows the access given for the interpreted object.

1.22.1. Reading interpretations

To read interpretations we will interact with the *api/interpretations* resource. The output in JSON response format could look like below (use e.g. *api/interpretations.json*):

```
{
  "interpretations": [{
    "created": "2013-10-07T11:37:19.273+0000",
    "lastUpdated": "2013-10-07T12:08:58.028+0000",
    "type": "map",
    "href": "https://play.dhis2.org/demo/api/interpretations/d3BukolfFZI",
    "id": "d3BukolfFZI"
  }, {
    "created": "2013-05-30T10:24:06.181+0000",
    "lastUpdated": "2013-05-30T10:25:08.066+0000",
    "type": "reportTable",
    "href": "https://play.dhis2.org/demo/api/interpretations/XSHiFlHAhhh",
    "id": "XSHiFlHAhhh"
  }, {
    "created": "2013-05-29T14:47:13.081+0000",
    "lastUpdated": "2013-05-29T14:47:13.081+0000",
    "type": "chart",
    "href": "https://play.dhis2.org/demo/api/interpretations/kr4AnZmYL43",
    "id": "kr4AnZmYL43"
  }
  ]
}
```

An interpretation contains properties for identifier, date of creation and date of last modification. The type property refers to the kind of object is being interpreted, and is useful to show an appropriate visual clue in a client. Valid options are "chart", "map", "reportTable" and "dataSetReport". By following the link given in the "href" property one can get more information about a specific interpretation. In the case of the map interpretation, the response will look like this:

```
{
  "created": "2013-10-07T11:37:19.273+0000",
  "lastUpdated": "2014-10-07T12:08:58.028+0000",
  "map": {
```

```

    "name": "ANC: ANC 2 Coverage",
    "created": "2014-11-13T12:01:21.918+0000",
    "lastUpdated": "2014-11-13T12:01:21.918+0000",
    "href": "https://play.dhis2.org/demo/api/maps/bhmHJ4ZCdCd",
    "id": "bhmHJ4ZCdCd"
  },
  "text": "We can see that the ANC 2 coverage of Kasonko and Lei districts are under 40 %. What could be the cause for this?",
  "comments": [{
    "created": "2014-10-07T12:08:58.026+0000",
    "lastUpdated": "2014-10-07T12:08:58.026+0000",
    "text": "Due to the rural environment, getting women to the facilities is a challenge. Outreach campaigns might be helpful.",
    "href": "https://play.dhis2.org/demo/api/null/iB4Etq8yTE6",
    "id": "iB4Etq8yTE6"
  }],
  "type": "map",
  "href": "https://play.dhis2.org/demo/api/interpretations/d3BukolfFZI",
  "id": "d3BukolfFZI"
}

```

The map interpretation contains identifier and type information in the "id" and "type" properties. The interpretation text is available in the "text" property and references to any comments in the "comments" list. It also contains information about the interpreted object, in this case the "map" property. Note that you can follow the link to the actual map through the "href" property. 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/maps/bhmHJ4ZCdCd/data
```

1.22.2. Writing interpretations

We will start by writing an interpretation for the chart with identifier *EbRN2VibPdV*. To write chart interpretations we will interact with the <http://play.dhis2.org/demo/api/interpretations/chart/{chartId}> resource. The interpretation will be the request body. Based on this we can put together the following request using cURL:

```
curl -d "This chart shows a significant ANC 1-3 dropout" "https://play.dhis2.org/demo/api/interpretations/chart/EbRN2VibPdV" \
-H "Content-Type:text/plain" -u admin:district -v
```

Second we will write a comment on the interpretation we just wrote. 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 <http://play.dhis2.org/demo/api/interpretations/{interpretationId}/comment> like this:

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

You can also write interpretations for report tables in a similar way by interacting with the <http://app.dhis2.org/demo/api/interpretations/reportTable/{reportTableId}>. For report tables you can also provide an optional *ou* query parameter to supply an organisation unit identifier in the case where the report table has an organisation unit report parameter:

```
curl -d "This table reveals poor data quality" "https://play.dhis2.org/demo/api/interpretations/reportTable/xIWpSo5jjT1?ou=O6uvpzGd5pu"
-H "Content-Type:text/plain" -u admin:district -v
```

1.22.3. Creating, updating and removing interpretation comments

Creating comments to existing interpretations:

```
POST "plain-text comment" to /api/interpretations/ID/comments
```

Updating comments in existing interpretations:

```
PUT "plain-text comment" to /api/interpretations/ID/comments/ID
```

Removing comments in existing interpretations:

```
DELETE /api/interpretations/ID/comments/ID
```

1.23. Viewing analytical resource representations

DHIS 2 has several resources for data analysis. These resources include *charts*, *maps*, *reportTables*, *reports* and *documents*. By visiting these resources you will retrieve information about the resource. For instance, by navigating to *api/charts/R0DVGvXDUNP* the response will contain the name, last date of modification and so on for the chart. To retrieve the analytical representation, for instance a PNG representation of the chart, you can append */data* to all these resources. For instance, by visiting *api/charts/R0DVGvXDUNP/data* the system will return a PNG image of the chart.

Table 1.37. Analytical resources

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

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

Table 1.38. Data query parameters

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

Table 1.39. Query parameters for png / image types (charts, maps)

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

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

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

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

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

api/reports/OeJsA6K10tx/data.pdf
```

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

1.24. Plugins

DHIS 2 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.

1.24.1. 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 DHIS 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>
<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"}]}
        ],
      ]
    }
  </script>
</head>
</html>
```

```

        // All following options are optional
        showTotals: false,
        showSubTotals: false,
        hideEmptyRows: true,
        showHierarchy: true,
        displayDensity: "comfortable",
        fontSize: "large",
        digitGroupSeparator: "comma",
        legendSet: {id: "BtxOoQuLygl"}
    });
}
</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 DHIS 2 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 DHIS 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 DHIS 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 DHIS 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 properties are required: *el* and *url* (please refer to the table below). Now, if you want to refer to pre-defined tables already made inside DHIS it is sufficient to provide the additional *id* parameter. If you instead want to configure a pivot table dynamically you should omit the *id* parameter and provide data dimensions inside a *columns* array, a *rows* array and optionally a *filters* array instead.

A data dimension is defined as an object with a text property called *dimension*. This property accepts the following values: *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 1.40. 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 DHIS server
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 ancestors
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)
userOrgUnit	string / array	No		Organisation unit identifiers,

Param	Type	Required	Options (default first)	Description
				overrides organisation units associated with current 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/reportTables>.

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

  DHIS.getTable({
    url: base,
    el: "table2",
    columns: [
      {dimension: "de", items: [{id: "YtbsuPPo010"}, {id: "16byfWFUGaP"}]}
    ],
    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>.

1.24.2. 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 DHIS back-end into a Web page. To accomplish this we will use the DHIS 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.

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 DHIS 2 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 DHIS 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 DHIS 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 DHIS 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 properties are required: *el* and *url* (please refer to the table below). Now, if you want to refer to pre-defined charts already made inside DHIS it is sufficient to provide the additional *id* parameter. If you instead want to configure a chart dynamically you should omit the *id* parameter and provide data dimensions inside a *columns* array (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 1.41. Visualizer chart plug-in configuration

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

Param	Type	Required	Options (default first)	Description
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
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 current 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/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: "16byfWFUGaP"}]}
],
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>.

1.24.3. Embedding maps with the GIS map plug-in

In this example we will see how we can embed maps with data served from a DHIS 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.

```

<!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
      });
    });
  </script>

```

```

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 DHIS 2 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 DHIS 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 DHIS 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 DHIS installation.

```

Ext.onReady( function() {
  Ext.Ajax.request({
    url: base + "dhis-web-commons-security/login.action",
    method: "POST",
    params: { j_username: "portal", j_password: "Portal123" },
    success: setLinks
  });
});

```

Now let us have a look at the various options for the GIS plug-in. Two properties are required: *el* and *url* (please refer to the table below). Now, if you want to refer to pre-defined maps already made in the DHIS 2 GIS it is sufficient to provide the additional *id* parameter. If you instead want to configure a map dynamically you should omit the *id* parameter and provide *mapViews* (layers) instead. They should be configured with data dimensions inside a *columns* array, a *rows* array and optionally a *filters* array instead.

A data dimension is defined as an object with a text property called *dimension*. This property accepts the following values: *in* (indicator), *de* (data element), *ds* (data set), *dc* (data element operand), *pe* (period), *ou* (organisation unit) or

the id of any organisation unit group set or data element group set (can be found in the web api). The data dimension also has an array property called *items* which accepts objects with an *id* property.

To sum up, if you want to have a layer with e.g. "ANC 1 Coverage" in your map you can make the following *columns* config:

```
columns: [{
  dimension: "in", // could be "in", "de", "ds", "dc", "pe", "ou" or any dimension id
  items: [{id: "Uvn6LCg7dVU"}], // the id of ANC 1 Coverage
}]
```

Table 1.42. 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 DHIS server
id	string	No		Identifier of a pre-defined map (favorite) in DHIS
baseLayer	string/boolean	No	'gs', 'googlestreets' 'gh', 'googlehybrid' 'osm', 'openstreetmap' false, null, 'none', 'off'	Show background map
hideLegend	boolean	No	false true	Hide legend panel
mapViews	array	Yes (if no id provided)		Array of layers

If no id is provided you must add map view objects with the following config options:

Table 1.43. Map plug-in configuration

layer	string	No	"thematic1" "thematic2" "thematic3" "thematic4" "boundary" "facility"	The layer to which the map view content should be added
columns	array	Yes		Indicator, data element, data operand or data set (only one will be used)
rows	array	Yes		Organisation units (multiple allowed)
filter	array	Yes		Period (only one will be used)

classes	integer	No	5 1-7	The number of automatic legend classes
method	integer	No	2 3	Legend calculation method where 2 = equal intervals and 3 = equal counts
colorLow	string	No	"ff0000" (red) Any hex color	The color representing the first automatic legend class
colorHigh	string	No	"00ff00" (green) Any hex color	The color representing the last automatic legend class
radiusLow	integer	No	5 Any integer	Only applies for facilities (points) - radius of the point with lowest value
radiusHigh	integer	No	15 Any integer	Only applies for facilities (points) - radius of the point with highest value
opacity	double	No	0.8 0 - 1	Opacity/transparency of the layer content
legendSet	object	No		Pre-defined legend set. Will override the automatic legend set.
labels	boolean/object	No	false true object properties: fontSize (integer), color (hex string), strong (boolean), italic (boolean)	Show labels on the map
width	integer	No		Width of map
height	integer	No		Height of map
userOrgUnit	string / array	No		Organisation unit identifiers, overrides organisation units associated with current user, single or array

We continue by adding one pre-defined and one dynamically configured map to our HTML document. You can browse the list of available maps using the Web API here: <http://play.dhis2.org/demo/api/maps>.

```
function setLinks() {
  DHIS.getMap({ url: base, el: "map1", id: "ytkZY3ChM6J" });

  DHIS.getMap({
```



```

url: base,
el: "map2",
mapViews: [
  columns: [ // Chart series
    columns: [{dimension: "in", items: [{id: "Uvn6LCg7dVU"}]}], // data
  ],
  rows: [ // Chart categories
    rows: [{dimension: "ou", items: [{id: "LEVEL-3"}, {id: "ImspTQPwCqd"}]}], //
organisation units
  ],
  filters: [
    filters: [{dimension: "pe", items: [{id: "LAST_3_MONTHS"}]}], // period
  ],
  // All following options are optional
  classes: 7,
  colorLow: "02079c",
  colorHigh: "e5ecff",
  opacity: 0.9,
  legendSet: {id: "fqs276KXCXi"}
]
});
}

```

Finally we include some *div* elements in the body section of the HTML document with the identifiers referred to in the plug-in Javascript.

```

<div id="map1"></div>
<div id="map2"></div>

```

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

1.24.4. 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 DHIS 2 demo site, make sure you update them to point to your own DHIS 2 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: 'jlgNXBgwKVm', 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>.

1.25. SQL views

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, let's 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/sqlViews/dI68mLkPlwN/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/sqlViews/{id}/data(.csv)

```

The *{server}* parameter should be replaced with your own server. The next part of the URL `/api/sqlViews/` should be appended with the specific SQL view identifier. Append either *data* for XML data or *data.csv* for comma delimited values. Support response formats are json, xml, csv, xls, html and html+css. As an example, the following command would retrieve XML data for the SQL view defined above.

```

curl "https://play.dhis2.org/demo/api/sqlViews/dI68mLkPlwN/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.

1.25.1. 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/sqlViews/{id}/data?criteria=col1:value1&criteria=col2:value2
```

As an example, to filter the SQL view result set above to only return organisation units at level 4 you can use the following URL:

```
https://play.dhis2.org/demo/api/sqlViews/dI68mLkPlwN/data.csv?criteria=level:4
```

1.25.2. Variables

SQL views support variable substitution. Variable substitution is only available for SQL view of type *query*, meaning SQL views which are not created in the database but simply executed as regular SQL queries. Variables can be inserted directly into the SQL query and must be on this format:

```
${variable-key}
```

As an example, an SQL query that retrieves all data elements of a given value type where the value type is defined through a variable can look like this:

```
select * from dataelement where valuetype = '${valueType}';
```

These variables can then be supplied as part of the URL when requested through the *sqlViews* Web API resource. Variables can be supplied on the following format:

```
/api/sqlViews/{id}/data?var=key1:value1&var=key2:value2
```

An example query corresponding to the example above can look like this:

```
/api/sqlViews/dI68mLkPlwN/data.json?var=valueType:int
```

The *valueType* variable will be substituted with the *int* value, and the query will return data elements with int value type.

The variable parameter must contain alphanumeric characters only. The variables must contain alphanumeric, dash, underscore and whitespace characters only.

1.26. 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.

1.26.1. 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/dashboards.json
```

The dashboards resource will provide a list of dashboards. Remember that the dashboard object is shared so the list will be affected by the currently authenticated user. You can retrieve more information about a specific dashboard by following its link, similar to this:

```
api/dashboards/vQFhmLJU5sK.json
```

A dashboard contains information like name and creation date and an array of dashboard items. The response in JSON format will look similar to this response (certain information has been removed for the sake of brevity).

```
{
  "lastUpdated" : "2013-10-15T18:17:34.084+0000",
  "id" : "vQFhmLJU5sK",
  "created" : "2013-09-08T20:55:58.060+0000",
  "name" : "Mother and Child Health",
  "href" : "https://play.dhis2.org/demo/api/dashboards/vQFhmLJU5sK",
  "publicAccess" : "-----",
  "externalAccess" : false,
  "itemCount" : 17,
  "displayName" : "Mother and Child Health",
  "access" : {
    "update" : true,
    "externalize" : true,
    "delete" : true,
    "write" : true,
    "read" : true,
    "manage" : true
  },
  "user" : {
    "id" : "xE7jOejl9FI",
    "name" : "John Traore",
    "created" : "2013-04-18T15:15:08.407+0000",
    "lastUpdated" : "2014-12-05T03:50:04.148+0000",
    "href" : "https://play.dhis2.org/demo/api/users/xE7jOejl9FI"
  },
  "dashboardItems" : [{
    "id" : "bulIANPFa9H",
    "created" : "2013-09-09T12:12:58.095+0000",
    "lastUpdated" : "2013-09-09T12:12:58.095+0000"
  }, {
    "id" : "ppFEJmWWDa1",
    "created" : "2013-09-10T13:57:02.480+0000",
    "lastUpdated" : "2013-09-10T13:57:02.480+0000"
  }
],
  "userGroupAccesses" : []
}
```

A more tailored response can be obtained by specifying specific fields in the request. An example is provided below, which would return more detailed information about each object on a users dashboard.

```
api/dashboards/vQFhmLJU5sK/?fields=:all,dashboardItems[:all]
```

1.26.2. 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/dashboards/q/my-query.json
```

JSON and XML response formats are currently 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": "LW0027b7TdD"
}, {
  "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
}

```

1.26.3. 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/dashboards/vQFhmLJU5sK
```

1.26.4. 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 1.44. 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
id	Identifier of the resource to be represented by the dashboard item	Resource identifier

A *POST* request URL for adding a chart to a specific dashboard could look like this, where the last id query parameter value is the chart resource identifier:

```
/api/dashboards/vQFhmLJU5sK/items/content?type=chart&id=LW0027b7TdD
```

When adding resource of type map, chart and report table, 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 the resource to it.

In order to move a dashboard item to a new position within the list of items in a dashboard, a consumer can make a *POST* request to the following resource URL, where <dashboard-id> should be replaced by the identifier of the dashboard, <item-id> should be replaced by the identifier of the dashboard item and <index> should be replaced by the new position of the item in the dashboard, where the index is zero-based:

```
/api/dashboards/<dashboard-id>/items/<item-id>/position/<index>
```

To remove a dashboard item completely from a specific dashboard a consumer can make a *DELETE* request to the below resource URL, where <dashboard-id> should be replaced by the identifier of the dashboard and <item-id> should be replaced by the identifier of the dashboard item. The dashboard item identifiers can be retrieved through a GET request to the dashboard resource URL.

```
/api/dashboards/<dashboard-id>/items/<item-id>
```

To remove a specific content resource within a dashboard item a consumer can make a *DELETE* request to the below resource URL, where <content-resource-id> should be replaced by the identifier of a resource associated with the dashboard item; e.g. the identifier of a report or a user. For instance, this can be used to remove a single report from a dashboard item of type reports, as opposed to removing the dashboard item completely:

```
/api/dashboards/<dashboard-id>/items/<item-id>/content/<content-resource-id>
```

1.27. Analytics

To access analytical, aggregated data in DHIS 2 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.

1.27.1. Dimensions and items

DHIS 2 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 DHIS 2. Each data dimension has a corresponding *dimension identifier*, and each dimension can have a set of *dimension items*:

Table 1.45. Dimensions and dimension items

Dimension	Dimension id	Dimension items
Data elements, indicators, data set reporting rates, data element operands, program indicators, program data elements, program attributes	dx	Data element, indicator, data set, data element operand, program indicator, program attribute identifiers, keyword DE_GROUP-<group-id>, use <dataelement-id>-<optioncombo-id> for operands
Periods (time)	pe	ISO periods and relative periods, see "date and period format"
Organisation unit hierarchy	ou	Organisation unit identifiers, and keywords USER_ORGUNIT, USER_ORGUNIT_CHILDREN, USER_ORGUNIT_GRANDCHILDREN, LEVEL-<level> and OU_GROUP-<group-id>
Category option combinations	co	Category option combo identifiers (omit to get all items)
Attribute option combinations	ao	Category option combo identifiers (omit to get all items)
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/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/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/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
&dimension=pe:2014Q1;2014Q2&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/analytics?dimension=dx:fbfJHSPpUQD;cYeuwXTCPkU
&dimension=co&dimension=pe:2014Q1&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/analytics?dimension=dx:DE_GROUP-h9cuJOkOwY2
&dimension=pe:2014Q1&dimension=ou:O6uvpzGd5pu
```

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/analytics?dimension=Bpx0589u8y0:oRVt7g429ZO;MAS88nJc9nL
```

```
&dimension=pe:2014&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/analytics?dimension=dx:s46m5MS0hxu;fClA2Erf6IO&dimension=pe:2014
&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/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:

```
api/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2014&dimension=ou:LEVEL-3
```

```
api/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2014
&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/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2014
&dimension=ou:OU_GROUP-w0gFTTmsUcF;O6uvpzGd5pu
```

```
api/analytics?dimension=dx:fbfJHSPpUQD&dimension=pe:2014
&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 and data sets 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.

1.27.2. Request query parameters

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

Table 1.46. Query parameters

Query parameter	Required	Description	Options
dimension	Yes	Dimensions to be retrieved, repeated for each.	Any dimension
filter	No	Filters to apply to the query, repeated for each.	Any dimension
aggregationType	No	Aggregation type to use in the aggregation process.	SUM AVERAGE_INT AVERAGE_INT_DISAGGREGATION AVERAGE_BOOL 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
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
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
approvalLevel	No	Include data which has been approved at least up to the given approval level, refers to identifier of approval level.	Identifier of approval level
relativePeriodDate	No	Date used as basis for relative periods.	Date.
userOrgUnit	No	Explicitly define the user org units to utilize, overrides organisation units associated with 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/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/analytics?
dimension=dx:fbfJHSPpUQD&dimension=pe:2014Q1&dimension=ou:O6uvpzGd5pu&aggregationType=COUNT
```

The *measureCriteria* query parameter lets you filter out ranges of data records to return. You can instruct the system to return only records where the aggregated data value is equal, greater than, greater or equal, less than or less or equal to certain values. You can specify any number of criteria on the following format, where *criteria* and *value* should be substituted with real values:

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

1.27.3. 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/analytics.xml?dimension=dx:fbfJHSPpUQD
```

```
&dimension=pe:2014&dimension=ou:O6uwpzGd5pu;lc3eMKXaEfw
```

The analytics responses must be retrieved using the HTTP *GET* method. This allows for direct linking to analytics responses from Web pages as well as other HTTP-enabled clients. To do functional testing we can use the cURL library. By executing this command against the demo database you will get an analytics response in JSON format:

```
curl "play.dhis2.org/demo/api/analytics.json?
dimension=dx:eTDtyyaSA7f;FbKK4ofIv5R&dimension=pe:2014Q1;2014Q2&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": [
      "2014Q1",
      "2014Q2"
    ],
    "ou": [
      "ImspTQPwCqd"
    ],
    "names": {
      "2014Q1": "Jan to Mar 2014",
      "2014Q2": "Apr to Jun 2014",
      "FbKK4ofIv5R": "Measles Coverage <1 y",
      "ImspTQPwCqd": "Sierra Leone",
      "eTDtyyaSA7f": "Fully Immunized Coverage"
    }
  },
  "rows": [
    [
      "eTDtyyaSA7f",
      "2014Q2",
      "81.1"
    ],
    [
      "eTDtyyaSA7f",
      "2014Q1",
      "74.7"
    ],
    [
      "FbKK4ofIv5R",
      "2014Q2",
```

```

        "88.9"
      ],
      [
        "FbKK4ofIv5R",
        "2014Q1",
        "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 aggregated data values.

1.27.4. 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.
- Indicators, data sets and categories cannot be specified as filters.
- 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.

1.27.5. 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/analytics/debug/sql?
dimension=dx:fbfJHSPpUQD;cYeuwXTCpkU&filter=pe:2014Q1;2014Q2&filter=ou:O6uvpzGd5pu;lc3eMKXaEfw
```

1.28. Event analytics

The event analytics API lets you access aggregated event data and query *events* captured in DHIS 2. 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.

1.28.1. 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 1.47. 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
Organisation unit group sets	<id>	Organisation unit group set identifiers

1.28.2. Request query parameters

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

Table 1.48. 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	

Query parameter	Required	Description	Options
		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 1.49. Query parameters for event query analytics only

Query parameter	Required	Description	Options
ouMode	No	The mode of selecting organisation units. Default is DESCENDANTS, meaning all sub units in the hierarchy. CHILDREN refers to immediate children in the hierarchy; SELECTED refers to the selected organisation units only.	DESCENDANTS, CHILDREN, SELECTED
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 1.50. 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

Query parameter	Required	Description	Options
collapseDataDimension	No	Collapse all data dimensions (data elements and attributes) into a single dimension in the response.	false true
skipMeta	No	Exclude the meta data part of the response (improves performance).	false true
skipData	No	Exclude the data part of the response.	false true
skipRounding	No	Skip rounding of aggregate data values.	false true
aggregateData	No	Produce aggregate values for the data dimensions (as opposed to dimension items).	false true

1.28.3. Event query analytics

The *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. 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/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 2014, where the "Gender" and "Age" data elements are included and the "Age" dimension is filtered on "18", you can use the following query:

```
api/analytics/events/query/eBAyeGv0exc?startDate=2014-01-01&endDate=2014-10-31
&dimension=ou:06uvpzGd5pu;fdc6uOvgoji&dimension=oZg33kd9taw&dimension=qrur9Dvnyt5:EQ:18
```

To retrieve events for the "Birth" program stage of the "Child programme" program between March and December 2014, where the "Weight" data element, filtered for values larger than 2000:

```
api/analytics/events/query/IpHINAT79UW?
stage=A03MvHHogjR&startDate=2014-03-01&endDate=2014-12-31
&dimension=ou:06uvpzGd5pu&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/analytics/events/query/eBAyeGv0exc?startDate=2014-01-01&endDate=2014-10-31
&dimension=ou:06uvpzGd5pu&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/analytics/events/query/eBAyeGv0exc?startDate=2014-01-01&endDate=2014-10-31
&dimension=ou:06uvpzGd5pu&dimension=qrur9Dvnyt5&page=3&pageSize=20
```

1.28.3.1. 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:

```
&dimension=qrur9Dvnyt5:GT:5;LT:15
```

The available operators are listed below.

Table 1.51. 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 ";"

1.28.3.2. 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/analytics/events/query/eBAyeGv0exc.xls?startDate=2014-01-01&endDate=2014-10-31
&dimension=ou:06uvpzGd5pu&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/analytics/events/query/eBAyeGv0exc?startDate=2014-01-01&endDate=2014-10-31
&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": "qrur9Dvny5",
    "column": "Age",
    "type": "java.lang.String",
    "hidden": false,
    "meta": false
  }
],
"metaData": {
  "names": {
    "qrur9Dvny5": "Age",
    "eBAyeGv0exc": "Inpatient morbidity and mortality",
    "ImspTQPwCqd": "Sierra Leone",
    "O6uvpzGd5pu": "Bo",
    "YuQRtpLP10I": "Badjia",
    "oZg33kd9taw": "Gender"
  },
  "ouHierarchy": {
    "YuQRtpLP10I": "/ImspTQPwCqd/O6uvpzGd5pu"
  },
  "width": 8,
  "height": 25,

```

```

    "rows": [
      [ "yx9IDINf82o", "Zj7UnCAulEk", "2014-08-05", "[5.12, 1.23]", "Ngelehun CHC",
        "OU_559", "YuQRtpLP10I", "Female", "50"],
      [ "IPNa7AsCyFt", "Zj7UnCAulEk", "2014-06-12", "[5.22, 1.43]", "Ngelehun CHC",
        "OU_559", "YuQRtpLP10I", "Female", "50"],
      [ "ZY9JL9dkhD2", "Zj7UnCAulEk", "2014-06-15", "[5.42, 1.33]", "Ngelehun CHC",
        "OU_559", "YuQRtpLP10I", "Female", "50"],
      [ "MYvh4WAUdWt", "Zj7UnCAulEk", "2014-06-16", "[5.32, 1.53]", "Ngelehun CHC",
        "OU_559", "YuQRtpLP10I", "Female", "50"]
    ]
  }

```

The *headers* section of the response describes the content of the query result. The event unique identifier, the program stage identifier, the event date, the organisation unit name, the organisation unit code and the organisation unit identifier appear as the first six dimensions in the response and will always be present. Next comes the data elements, person attributes and person identifiers which were specified as dimensions in the request, in this case the "Gender" and "Age" data element dimensions. The header section contains the identifier of the dimension item in the "name" property and a readable dimension description in the "column" property.

The *metaData* section, *ou* object contains the identifiers of all organisation units present in the response mapped to a string representing the hierarchy. This hierarchy string lists the identifiers of the ancestors (parents) of the organisation unit starting from the root. The *names* object contains the identifiers of all items in the response mapped to their names.

The *rows* section contains the events produced by the query. Each row represents exactly one event.

1.28.4. Event aggregate analytics

In order to get *aggregated numbers* of events captured in DHIS 2 you can work with the *analytics/events/aggregate* resource. This resource lets you retrieve aggregate data based on a program and optionally a program stage, and lets you filter on any event dimension. In other words, it 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/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 2014, 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/analytics/events/aggregate/eBAyeGv0exc?startDate=2014-01-01&endDate=2014-10-31
&dimension=ou:06uvpzGd5pu&dimension=oZg33kd9taw:EQ:Female&dimension=qrur9Dvny5:GT:50

```

To retrieve data for fixed and relative periods instead of start and end date, in this case May 2014 and last 12 months, and the organisation unit associated with the current user, you can use the following query:

```

api/analytics/events/aggregate/eBAyeGv0exc?dimension=pe:201405;LAST_12_MONTHS
&dimension=ou:USER_ORGUNIT;fdc6uOvg07ji&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/analytics/events/aggregate/eBAyeGv0exc?dimension=pe:2014;
&dimension=ou:06uvpzGd5pu&filter=oZg33kd9taw:EQ:Female

```

To specify the "Bo" organisation unit and the period "2014" 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/analytics/events/aggregate/eBAyeGv0exc?filter=pe:2014&filter=ou:06uvpzGd5pu

```

```
&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/analytics/events/aggregate/eBAyeGv0exc?filter=pe:2014&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/analytics/events/aggregate/eBAyeGv0exc.json?
stage=Zj7UnCAulek&dimension=ou:ImspTQPwCqd
&dimension=pe:LAST_12_MONTHS&dimension=fWIAEtYVEGk&value=qrur9Dvnyt5&aggregationType=AVERAGE
```

1.28.4.1. 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/analytics/events/aggregate/eBAyeGv0exc.json?
stage=Zj7UnCAulek&dimension=qrur9Dvnyt5-Yf6UHoPkdS6
&dimension=ou:ImspTQPwCqd&dimension=pe:LAST_12_MONTHS
```

1.28.4.2. 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:

```
{
  "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",
        "201405",
        "O6uwpzGd5pu",
        "2"
    ],
    [
        "Female",
        "63",
        "201405",
        "O6uwpzGd5pu",
        "2"
    ],
    [
        "Female",
        "67",
        "201405",
        "O6uwpzGd5pu",
        "1"
    ],
    [
        "Female",
        "71",
        "201405",
        "O6uwpzGd5pu",
        "1"
    ],
    [
        "Female",
        "75",
        "201405",
        "O6uwpzGd5pu",
        "14"
    ],
    [
        "Female",
        "73",
        "201405",
        "O6uwpzGd5pu",
        "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.

1.29. Geo features

The *geoFeatures* resource lets you retrieve geospatial information from DHIS 2. Geo features are stored together with organisation units, and 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/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/geoFeatures.json?ou=ou:LEVEL-4;O6uwpzGd5pu
```

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

Table 1.52. 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

1.29.1. GeoJSON

Support for GeoJSON output was added in 2.17, 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+geosjon*.

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/organisationUnits.geosjon?level=2&level=4
```

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

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

1.30. Generating resource and analytics tables

DHIS 2 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 DHIS 2 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/resourceTables
```

The analytics tables are optimized for data aggregation and used currently in DHIS 2 for the pivot table module. The analytics tables can be generated with a POST or PUT request to:

```
api/resourceTables/analytics
```

Table 1.53. 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.

1.31. 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.

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

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

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

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

Drop SQL views will drop all SQL views in the database. Note that it will not delete the DHIS 2 SQL views.

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

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

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

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

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

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

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

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

```
/api/maintenance/ouPathsUpdate
```

1.32. System resource

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

1.32.1. Generate identifiers

To generate valid, random DHIS 2 identifiers you can do a GET request to this resource:

```
http://<server-url>/api/system/id?n=3
```

The *n* 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",
    "WIOVHXuWQuV",
    "BRJNBpu4ki"
  ]
}
```

The DHIS 2 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).

1.32.2. View system information

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

```
http://yourdomain.com/api/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 resource 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 (KHTML, like Gecko) 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"
  },
  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:

```
http://yourdomain.com/api/system/context
```

1.32.3. 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*:

```
http://<server-url>/api/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 1.54. 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

1.32.4. 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 supplying 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 1.55. Task categories

Identifier	Description
ANALYTICSTABLE_UPDATE	Update of the analytics tables.
RESOURCESTABLE_UPDATE	Update 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/system/tasks/{task-category-id}
```

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

```
api/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
  }
]
```

1.32.5. Get appearance information

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

```
api/system/flags
```

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

```
api/system/styles
```

1.33. Users

This section covers the user resource methods.

1.33.1. 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 1.56. 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.
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/users?query=konan&authSubset=true&pageSize=10
```

1.33.2. 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/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/users/ID
```

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

1.33.3. 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/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/users/invites" -H "Content-Type:application/json" -u admin:district -v
```

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

1.33.4. 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/users/<uid>/replica
```

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

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

1.34. 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 relevant to the current user:

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

1.35. 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/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/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 -value pair for each system setting key-value pair using a *POST* request:

```
{
  "keyApplicationNotification": "Welcome",
  "keyApplicationIntro": "DHIS 2",
  "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/systemSettings/my-key
```

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

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

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

```
curl "play.dhis2.org/demo/api/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/systemSettings
```

To delete a system setting, you can make a *DELETE* request to the URL similar to the one used above for retrieval.

1.36. 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 save or update a setting for the currently authenticated user you can make a *POST* request to the following URL:

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

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

```
/api/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 "play.dhis2.org/demo/api/userSettings/my-key" -d "My long value" -H "Content-Type: text/plain" -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/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.

1.37. Organisation units

The organisationUnits resource follows the standard conventions as other metadata resources in DHIS 2. This resource supports some additional query parameters.

Table 1.57. Organisation units query parameters

Query parameter	Description
userOnly	Data capture organisation units associated with current user only.
userDataViewOnly	Data view organisation units associated with current user only.
userDataViewFallback	Data view organisation units associated with current user only with fallback to data capture organisation units.
level	Organisation units at the given level in the hierarchy.
maxLevel	Organisation units at the given max level or levels higher up in the hierarchy.

1.38. Static content

The *staticContent* resource allows you to upload and retrieve custom logos used in DHIS2. The resource lets the user upload a file with an associated key, which can later be retrieved using the key. Only PNG files are supported and can only be uploaded to the "logo_banner" and "logo_front" keys.

To upload a file, send the file with a *POST* request to:

```
POST /api/staticContent/<key>
```

Example request to upload logo.png to the logo_front key:

```
curl -F "file=@logo.png;type=image/png" "aps.dhis2.org/demo/api/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/staticContent/<key>
```

Example request to retrieve the file stored for logo_front:

```
curl "aps.dhis2.org/demo/api/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.

1.39. 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/configuration
```

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

```
/api/configuration/systemId
```

```
/api/configuration/feedbackRecipients
```

```
GET /api/configuration/offlineOrganisationUnitLevel
```

```
GET /api/configuration/infrastructuralDataElements
```

```
GET /api/configuration/infrastructuralIndicators
```

```
GET /api/configuration/infrastructuralPeriodType
```

```
GET /api/configuration/selfRegistrationRole
```

```
GET /api/configuration/selfRegistrationOrgUnit
```

```
GET /api/configuration/remoteServerUrl
```

```
GET /api/configuration/remoteServerUsername
```

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" ]
```

```
/api/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 1.58. Configuration values

Configuration property	Value
feedbackRecipients	User group ID
offlineOrganisationUnitLevel	Organisation unit level ID
infrastructuralDataElements	Data element group ID
infrastructuralIndicators	Indicator group ID
infrastructuralPeriodType	Period type name (e.g. "Monthly")
selfRegistrationRole	User role ID
selfRegistrationOrgUnit	Organisation unit ID
smtpPassword	SMTP email server password
remoteServerUrl	URL to remote server
remoteServerUsername	Username for remote server authentication
remoteServerPassword	Password for remote server authentication

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

1.40. Internationalization

In order to retrieve key-value pairs for translated strings you can use the *i18n* resource. 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/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"
}
```

1.41. SVG conversion

The Web API provides a resource which can be used to convert SVG content into more widely used formats such as PNG and PDF. Ideally this conversion should happen on the client side, but not all client side technologies are capable of performing this task. Currently PNG and PDF output formats are supported. The SVG content itself should be passed with a *svg* query parameter, and an optional query parameter *filename* can be used to specify the filename of the response attachment file. Note that the file extension should be omitted. For PNG you can send a *POST* request to the following URL with Content-type *application/x-www-form-urlencoded*, identical to a regular HTML form submission.

```
api/svg.png
```

For PDF you can send a *POST* request to the following URL with Content-type *application/x-www-form-urlencoded*.

```
api/svg.pdf
```

Table 1.59. Query parameters

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

1.42. Tracked entity management

Tracked entity have full CRUD (create, read, update, delete) support in the Web-API. A tracked entity only have two required property, *name* and *description*.


```
{
  "name": "Name of tracked entity",
  "description": "Description of tracked entity"
}
```

This payload can be sent to the *trackedEntities* resource, both **POST** and **PUT** are supported. For deleting a tracked entity you must use the **DELETE** method at the */api/trackedEntities/UID* resource.

1.43. 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.

1.43.1. 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": "cyl5vuJ5ETQ",
  "orgUnit": "DiszpKrYNg8",
  "attributes": [
    {
      "attribute": "dv3nChNSIxy",
      "value": "Joe"
    },
    {
      "attribute": "hw1RTFIFSUq",
      "value": "Smith"
    }
  ]
}
```

To push this to the server you can use the cURL command like this:

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

1.43.2. 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>
```

1.43.3. Deleting a tracked entity instance

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

1.43.4. 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": "ZRyCnJlqUXS",
  "orgUnit": "ImspTQPwCqd",
  "program": "S8uo8AlvYMz",
  "dateOfEnrollment": "2013-09-17",
  "dateOfIncident": "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>
```

1.43.5. 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 1.60. 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
```

1.44. Enrollment instance query

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

1.44.1. Request syntax

Table 1.61. Tracked entity instances query parameters

Query parameter	Description
ou	Organisation unit identifiers, separated by ";".
ouMode	The mode of selecting organisation units, can be <code>SELECTED</code> <code>CHILDREN</code> <code>DESCENDANTS</code> <code>ACCESSIBLE</code> <code>ALL</code> . Default is <code>SELECTED</code> , 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 <code>ACTIVE</code> <code>COMPLETED</code> <code>CANCELLED</code> .
followUp	Follow up status of the instance for the given program. Can be <code>true</code> <code>false</code> or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
trackedEntityInstance	Tracked entity instance identifier. Should not be used together with <code>trackedEntity</code> .
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 1.62. Organisation unit selection modes

Mode	Description
<code>SELECTED</code>	Organisation units defined in the request (default).
<code>CHILDREN</code>	Immediate children, i.e. only the first level below, of the organisation units defined in the request.
<code>DESCENDANTS</code>	All children, i.e. at only levels below, e.g. including children of children, of the organisation units defined in the request.
<code>ACCESSIBLE</code>	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.
<code>ALL</code>	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/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/enrollments.json?ou=06uvpzGd5pu&ouMode=DESCENDANTS&program=urlEdk5Oe2n
```

To specify program enrollment dates as part of the query:

```
api/enrollments.json?
&ou=06uvpzGd5pu&program=urlEdk5Oe2n&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/enrollments.json?ou=06uvpzGd5pu&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/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/enrollments.json?ou=06uvpzGd5pu&ouMode=DESCENDANTS&page=2&pageSize=3
```

1.44.2. 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": "urlEdk5Oe2n",
      "enrollment": "HLFOK0XThjr",
      "trackedEntityInstance": "qv0j4JBXQX0",
      "followup": false,
      "dateOfEnrollment": "2013-05-23T05:27:48.490+0000",
      "dateOfIncident": "2013-05-10T05:27:48.490+0000",
      "status": "ACTIVE"
    }
  ]
}
```

1.45. Tracked entity instance query

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

1.45.1. Request syntax

Table 1.63. Tracked entity instances query parameters

Query parameter	Description
filter	Attributes to use as a filter for the query. Param can be repeated any number of times. Filters can be applied to a dimension on the format <attribute-id>:<operator>:<filter>[:<operator>:<filter>]. Filter values are case-insensitive and can be repeated together with operator any number of times. Operators can be EQ GT GE LT LE NE LIKE IN.
ou	Organisation unit identifiers, separated by ";".
ouMode	The mode of selecting organisation units, can be SELECTED CHILDREN DESCENDANTS ACCESSIBLE ALL. Default is SELECTED, which refers to the selected organisation units only. See table below for explanations.
program	Program identifier. Restricts instances to being enrolled in the given program.
programStatus	Status of the instance for the given program. Can be ACTIVE COMPLETED CANCELLED.
followUp	Follow up status of the instance for the given program. Can be true false or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
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 1.64. 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.

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 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=urlEdk5Oe2n&filter=lw1SqMlfnfh: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=urlEdk5Oe2n
```

To specify program enrollment dates as part of the query:

```
api/trackedEntityInstances.json?
filter=zHxD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu&program=urlEdk5Oe2n
&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 1.65. 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 ","

1.45.2. 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": "ueuQlqb8cc1",
      "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": "lw1SqMlfnfh",
  "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": "Qo57lyj6Zcn",
        "type": "string",
        "value": "-30.659626"
    },
    {
        "code": "Longitude",
        "displayName": "Longitude",
        "attribute": "RG7uGl4w5Jq",
        "type": "string",
        "value": "26.916172"
    }
]
}
]
}

```

1.46. 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.

1.46.1. Request syntax

Table 1.66. 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 <code><operator>:<query></code> . 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 <code><attribute-id>:<operator>:<filter>[:<operator>:<filter>]</code> . 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.

Query parameter	Description
filter	Attributes to use as a filter for the query. Param can be repeated any number of times. Filters can be applied to a dimension on the format <attribute-id>:<operator>:<filter>[:<operator>:<filter>]. Filter values are case-insensitive and can be repeated together with operator any number of times. Operators can be EQ GT GE LT LE NE LIKE IN.
ou	Organisation unit identifiers, separated by ";".
ouMode	The mode of selecting organisation units, can be SELECTED CHILDREN DESCENDANTS ACCESSIBLE ALL. Default is SELECTED, which refers to the selected organisation units only. See table below for explanations.
program	Program identifier. Restricts instances to being enrolled in the given program.
programStatus	Status of the instance for the given program. Can be ACTIVE COMPLETED CANCELLED.
followUp	Follow up status of the instance for the given program. Can be true false or omitted.
programStartDate	Start date of enrollment in the given program for the tracked entity instance.
programEndDate	End date of enrollment in the given program for the tracked entity instance.
trackedEntity	Tracked entity identifier. Restricts instances to the given tracked instance type.
eventStatus	Status of any event associated with the given program and the tracked entity instance. Can be ACTIVE COMPLETED VISITED SCHEDULED OVERDUE SKIPPED.
eventStartDate	Start date of event associated with the given program and event status.
eventEndDate	End date of event associated with the given program and event status.
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 1.67. 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/trackedEntityInstances/grid.json?ou=DiszpKrYNg8
```

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

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

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

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

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

```
api/trackedEntityInstances/grid.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/trackedEntityInstances/grid.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 used as a filter:

```
api/trackedEntityInstances/grid.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/trackedEntityInstances/grid.json?
ou=DiszpKrYNg8&program=urlEdk5Oe2n&filter=lw1SqMlfnfh:GT:150:LT:190
```

To query on an attribute using multiple values in an IN filter:

```
api/trackedEntityInstances/grid.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/trackedEntityInstances/grid.json?
filter=zHXD5Ve1Efw:EQ:A&ou=O6uvpzGd5pu&ouMode=DESCENDANTS&program=urlEdk5Oe2n
```

To specify program enrollment dates as part of the query:

```
api/trackedEntityInstances/grid.json?filter=zHxD5Ve1Efw:EQ:A
&ou=06uvpzGd5pu&program=urlEdk5Oe2n
&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/grid.json?attribute=zHxD5Ve1Efw:EQ:A
&ou=06uvpzGd5pu&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/grid.json?
attribute=zHxD5Ve1Efw:EQ:A&ou=06uvpzGd5pu&ouMode=DESCENDANTS&page=2&pageSize=3
```

To query for instances which have events of a given status within a given time span:

```
api/trackedEntityInstances/grid.json?ou=06uvpzGd5pu
&program=urlEdk5Oe2n&eventStatus=LATE_VISIT
&eventStartDate=2014-01-01&eventEndDate=2014-09-01
```

You can use a range of operators for the filtering:

Table 1.68. 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 ";"

1.46.2. 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": "zHXD5VelEfw",
    "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"]
]
}

```

1.47. 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 DHIS 2 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.

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

1.47.2. 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 DHIS 2 user account has a valid email address associated with it. You can send a test email like this:

```
curl "localhost/api/email/test" -X POST -H "Content-Type:application/json" -u admin:district -v
```

1.48. Sharing

The sharing solution allows you to share most objects in the system with specific user groups and to define whether objects should be public and private. To get and set sharing for objects you can interact with the *sharing* resource. To request the sharing status for an object use a GET request to:

```
api/sharing?type=dataElement&id=fbfJHSPpUQD
```

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:

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

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

1.49. 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/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/scheduling" -d @scheduling.json -X POST -u admin:district -H
"Content-Type:application/json" -v
```

The table below lists available task strategies.

Table 1.69. Task strategies

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

1.50. Schema Resource

A new resource was included in DHIS 2.15 which can be used to introspect all available DXF2 classes, this resource can be found on `/api/schemas` and for a specific resource, you can have a look at `/api/schemas/TYPE`.

Example 1: Get all available schemas in XML:

```
GET /api/schemas.xml
```

Example 2: Get all available schemas in JSON:

```
GET /api/schemas.json
```

Example 3: Get JSON schema for a specific class:

```
GET /api/schemas/dataElement.json
```

1.51. UI customization with Javascript and CSS files

To customize the UI of the DHIS 2 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 DHIS 2 web application. This can be particularly useful in certain situations:

- Overriding the CSS styles of the DHIS 2 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.

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/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:

```
localhost/api/files/script
```

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

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

```
localhost/api/files/style
```

1.52. Synchronization

This section covers pull and push of data and metadata.

1.52.1. 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/synchronization/dataPush
```

1.52.2. Metadata pull

To initiate a metadata pull from a remote JSON document you can make a POST request with a *url* query parameter to the following resource:

```
api/synchronization/metadataPull?url=<url-to-json-document>
```

1.53. FRED API

DHIS 2 from version 2.11 implements support for the current draft of the FRED API version 1.0. The project defines itself as “open standard for sharing and updating health facility data”. The full specification, including representation format and basic usage, can be found at <http://facilityregistry.org/>.

Since version 1.0 is not finalized there are parts of the current specification that has not been implemented as we found it not to be in a stable enough state. Most notably we do not currently support sorting (we do however sort on name by default) and filtering of facilities.

The entry point for the implementation can be found at `http://<server-url>/api-fred` and the current version is located at `http://<server-url>/api-fred/v1`.

This section will give some simple examples of using the API.

Get all facilities:

```
curl -u username:password -X GET http://<server-url>/api-fred/v1/facilities.json
```

Get a specific facility based on either identifier or UUID:

```
curl -u username:password -X GET http://<server-url>/api-fred/v1/facilities/<id>.json
curl -u username:password -X GET http://<server-url>/api-fred/v1/facilities/<uuid>.json
```


Create a new facility:

```
curl -u username:password -X POST -d @new_facility.json
-H "Content-Type: application/json" http://<server-url>/api-fred/v1/facilities.json
```

Update a facility:

```
curl -u username:password -X POST -d @updated_facility.json
-H "Content-Type: application/json" http://<server-url>/api-fred/v1/facilities/
<id>.json

curl -u username:password -X POST -d @updated_facility.json
-H "Content-Type: application/json" http://<server-url>/api-fred/v1/facilities/
<uuid>.json
```

1.54. Data store

Using the *dataStore* resource, developers can store arbitrary data for their apps in a key-value structure. Access to a datastore is limited to the user's access to the corresponding app. For example a user with access to the "sampleApp" application will also be able to use the sampleApp namespace in the datastore.

1.54.1. Get keys and namespaces

For a list of all existing namespaces:

```
GET /api/dataStore
```

Example curl request for listing:

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

Example response:

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

For a list of all keys in a namespace:

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

Example curl request for listing:

```
curl "play.dhis2.org/demo/api/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/dataStore/<namespace>/<key>
```

Example curl request for retrieval:

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

Example response:

```
{
```

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

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

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

Example curl request for retrieval:

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

Example response:

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

1.54.2. Create and update values

To create a new key and value for a namespace:

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

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

```
curl "play.dhis2.org/demo/api/dataStore/foo/key_1" -X POST -d '{"foo":"bar"}' -u admin:district -v
```

Example response:

```
{
  "httpStatus": "OK",
  "httpStatusCode": 201,
  "status": "OK",
  "message": "Key 'key_1' created."
}
```

To update a key that exists in a namespace:

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

Example curl request for update, assuming valid JSON payload:

```
curl "play.dhis2.org/demo/api/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."
}
```

1.54.3. Delete keys

To delete an existing key from a namespace:

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

Example curl request for delete:

```
curl "play.dhis2.org/demo/api/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/dataStore/<namespace>
```

Example curl request for delete:

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

Example response:

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

1.55. Metadata repository

DHIS 2 provides a metadata repository containing metadata packages with various content. A metadata package is a

To retrieve an index over available metadata packages you can issue a GET request to the *metadataRepo* resource:

```
GET /api/metadataRepo
```

A metadata package entry contains information about the package and a URL to the relevant package. A metadata package is simply a DXF metadata file. 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://raw.githubusercontent.com/dhis2/dhis2-metadata-repo/master/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://raw.githubusercontent.com/dhis2/dhis2-metadata-repo/master/repo/221/trainingland-org-units/metadata.json"
  }
  ]
}
```

A client can follow the URLs in a RESTful fashion and install a metadata package by supplying POST request as text/plain with the metadata package URL as the payload to the *metadataPull* resource:

```
POST /api/metadataPull
```

An example curl command looks like this:

```
curl "localhost:8080/api/synchronization/metadataPull" -X POST
-d "https://raw.githubusercontent.com/dhis2/dhis2-metadata-repo/master/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 DHIS 2 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 thorough 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 DHIS 2, without the need to modify the source code of DHIS 2 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 DHIS 2 core functionality with custom solutions in a loosely coupled and clean manner.

Apps do not have permissions to interact directly with DHIS 2 Java API. Instead, apps are expected to use functionality and interact with the DHIS 2 services and data by utilizing the DHIS 2 Web API.

2.2. Creating Apps

DHIS 2 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",
  "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": "*"
    }
  }
}
```

The *manifest.webapp* file must be located at the root of the project. Among the properties, the *icons#48* property is used for the icon that is displayed on the list of apps that are installed on a DHIS 2 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 DHIS 2. The *** value for *href* is converted to the appropriate URL when the app is uploaded and installed in DHIS 2. This value can then be used by the application's JavaScript and HTML files to make calls to the DHIS 2 Web API and identify the correct location of DHIS 2 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://play.dhis2.org/demo"
```

```

    }
  }
}

```

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)

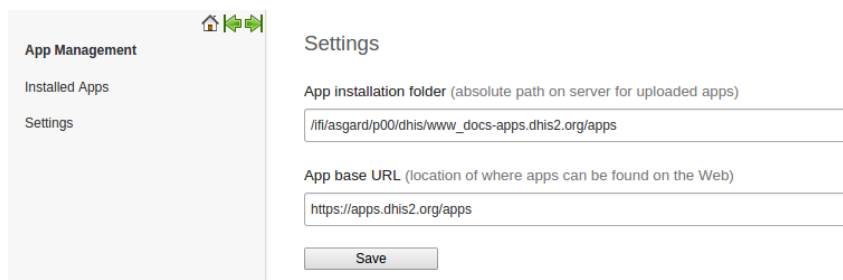
```

Note that it is only the `manifest.webapp` file which must be placed in the root. It is up to 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 DHIS 2 as you will see in the next section.

2.3. Configuring DHIS 2 for Apps Installation

The *App Manager* is found under Services # Apps. If your logged in user has permissions to view and edit settings you will see the Settings link in the left menu.

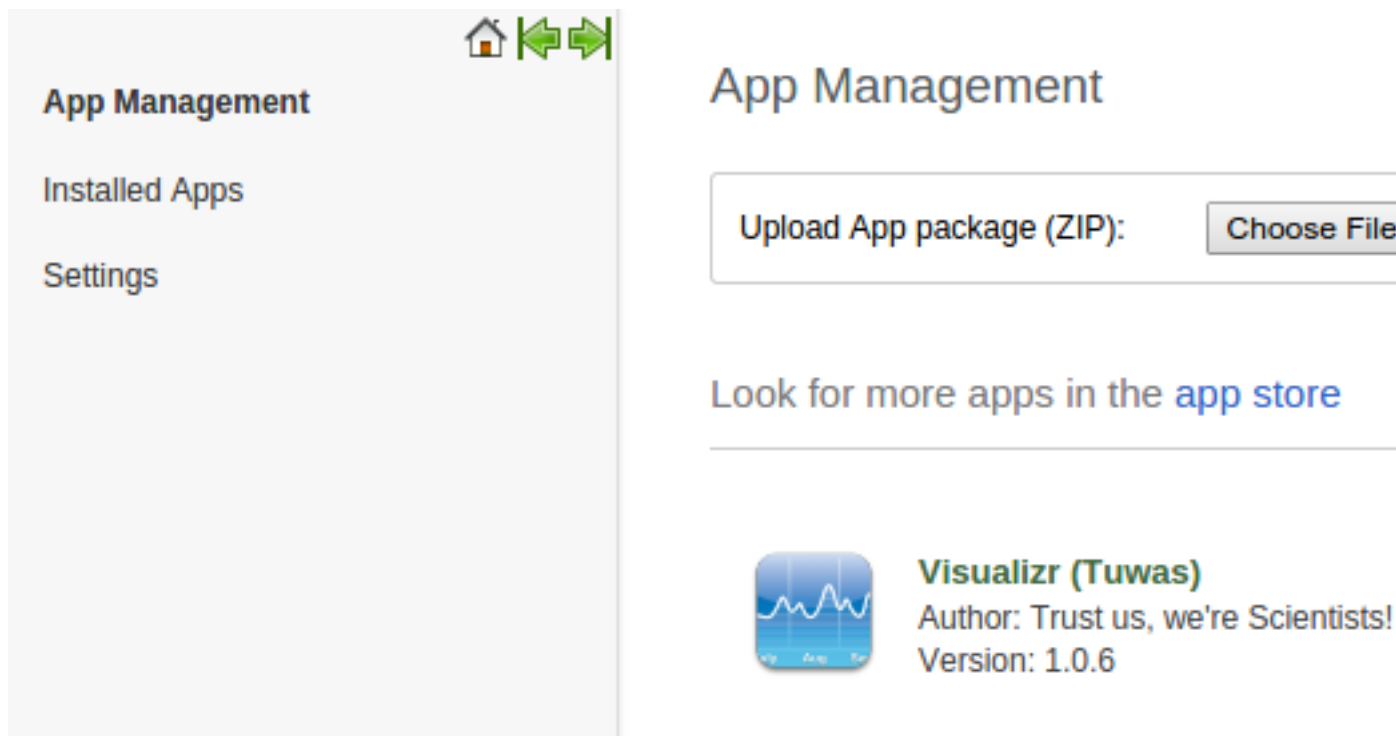


The following settings can be configured:

1. *App Installation Folder*: The folder on the file system where apps are unpacked. By default this is under the expanded DHIS folder suffixed by `/apps`. If you like to install your apps in another location, say `www` folder of Apache 2, you can specify the absolute path to that directory on the server, making your apps to be unpacked at that location.
2. *App Base URL*: The URL through which the apps can be found on the Web. By default this is the same as your DHIS 2 URL suffixed by `/apps`. If you are installing apps through a different web server you need to provide the full URL for that web server.

2.4. Installing Apps into DHIS 2

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 DHIS 2. You can also browse through apps in the DHIS 2 [app store](#) and download apps from there. The DHIS 2 app store allows for app searching, reviewing, commenting, requesting features, rating on the apps by the community.

2.5. 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.

2.6. Web-API for Apps

From version 2.14 there is also additional support for apps through the web-api. 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, excluding the file extension.

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/apps
```

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

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

To install an app, the following command can be issued:

```
curl -X POST -u user:pass -F file=@app.zip http://server.com/api/apps
```

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

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

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 DHIS 2 user interface.

```
curl -X PUT -u user:pass http://server.com/api/apps
```

To let DHIS 2 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/apps
```

To set the apps configuration you can make a POST request to the *config* resource 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/apps/config
```

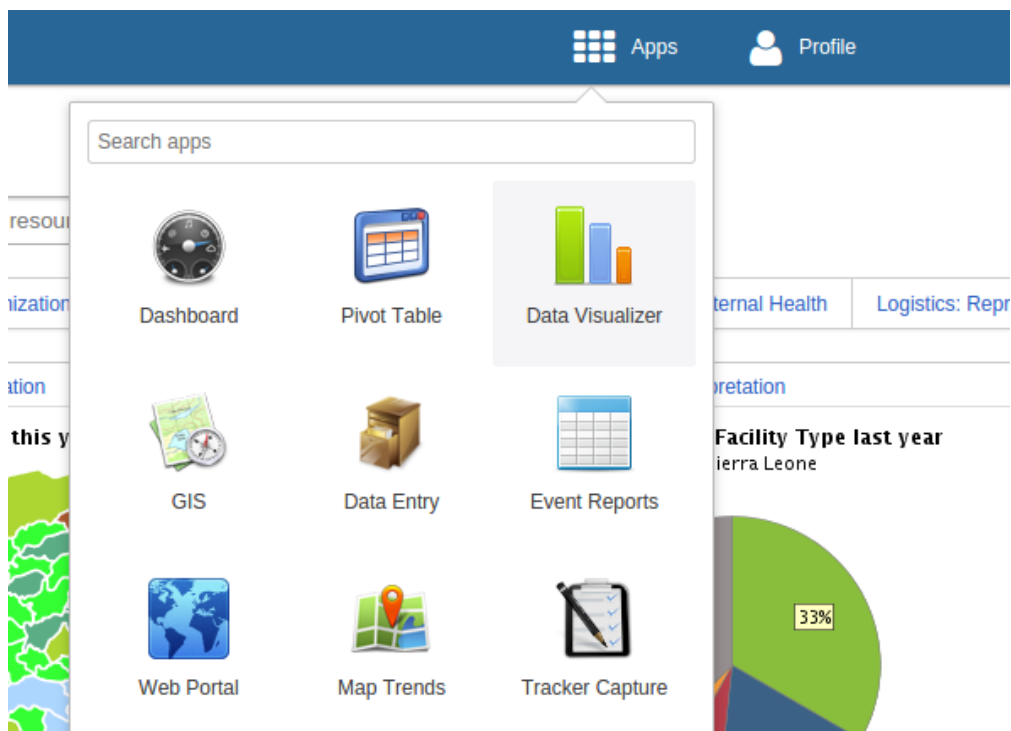
To restore the default app settings you can make a DELETE request to the config resource:

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

2.7. Adding the DHIS 2 menu to your app

In order to maintain a uniform appearance within DHIS 2 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 DHIS 2 user interface, where your app's icon will be placed.



The first step 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. DHIS 2 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 DHIS 2 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. Infrastructure

3.1. Release process

Checklist for release.

1. In order to tag the 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
```

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.
4. Enable email notifications for release branch.
5. Create Jenkins for build for the release WAR file.
6. Create automatic copy job from Jenkins to dhis2.org.
7. Create automatic update of play.dhis2.org/demo and play.dhis2.org/dev systems.
8. Update the database and WAR file on play.dhis2.org/demo and play.dhis2.org/dev instances. Run reinit-dhis-instance to make the changes take effect.
9. Create a new DHIS 2 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.

10. Create Javadoc with:

```
mvn javadoc:aggregate
```

The doc will be put in target folder. Zip it, upload to dhis2.org, unzip and place it in download 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. Write and send release email.

DHIS 2 Technical Architecture

1. Overview

This document outlines the technical architecture for the District Health Information Software 2 (DHIS 2). The DHIS 2 is a routine data based health information system which allows for data capture, aggregation, analysis, and reporting of data.

DHIS 2 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.

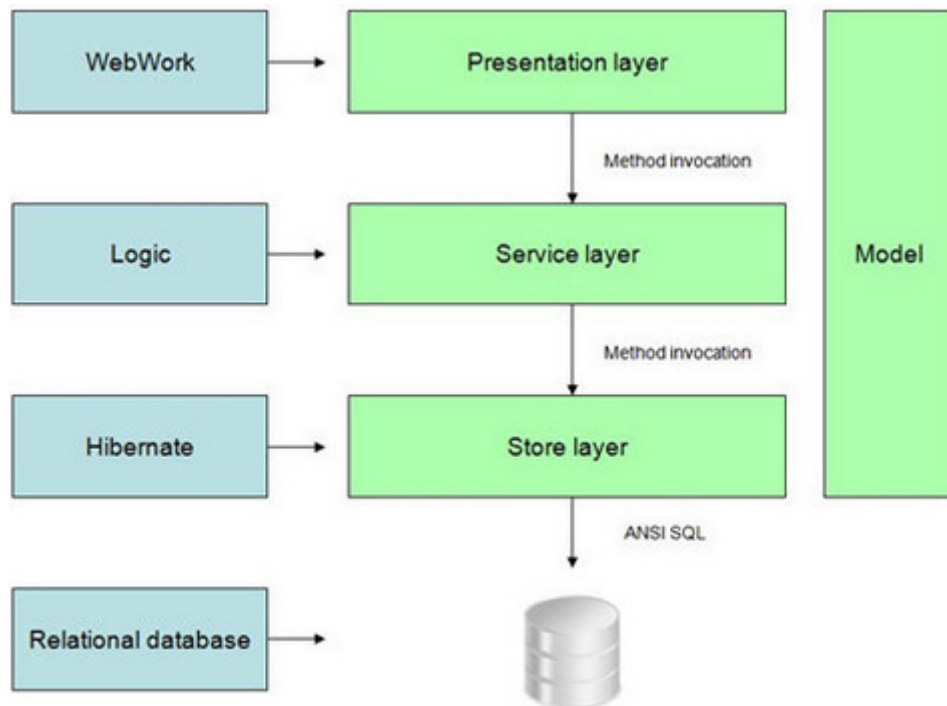


Fig. Overall architecture

2. Technical Requirements

The DHIS 2 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

3. Project Structure

DHIS 2 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.

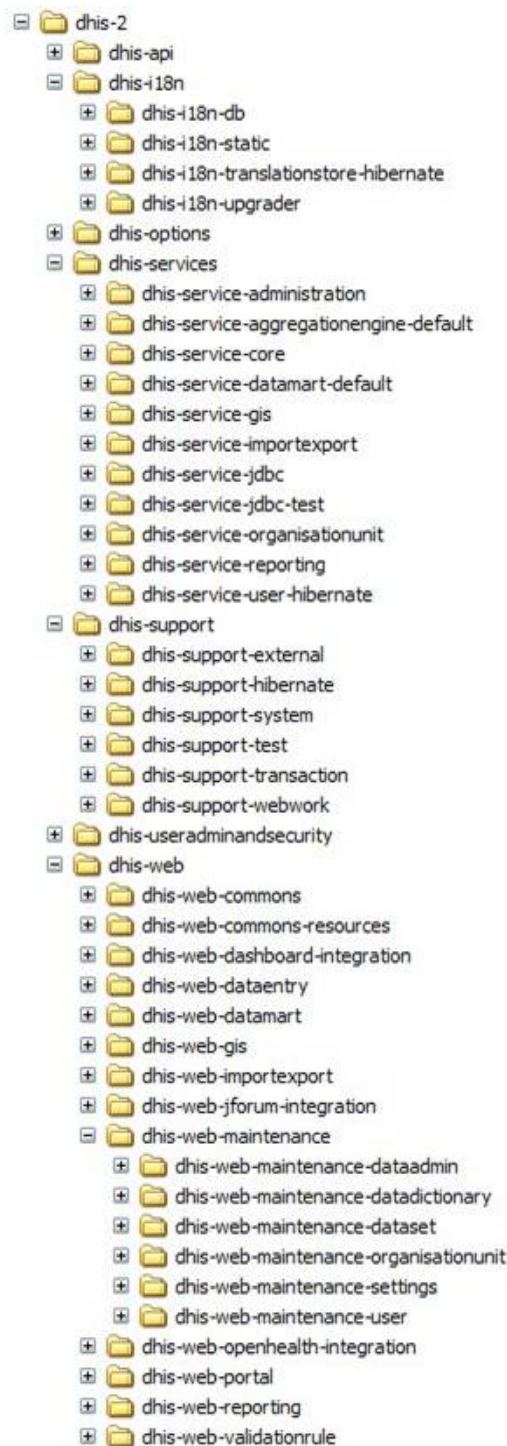
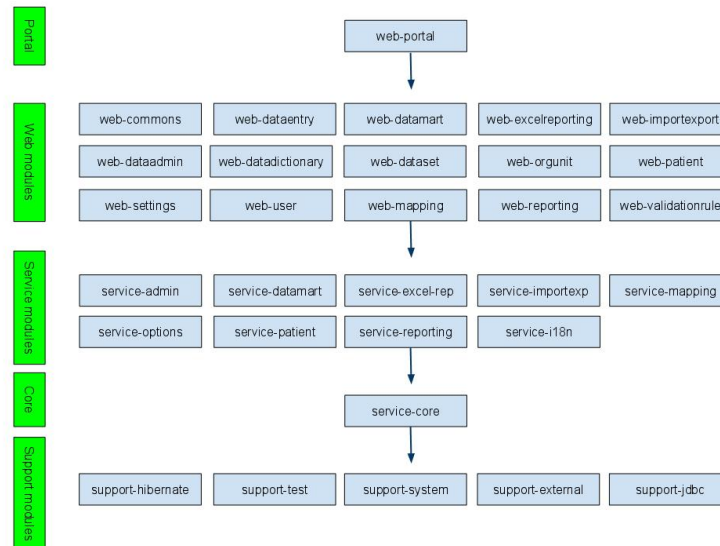


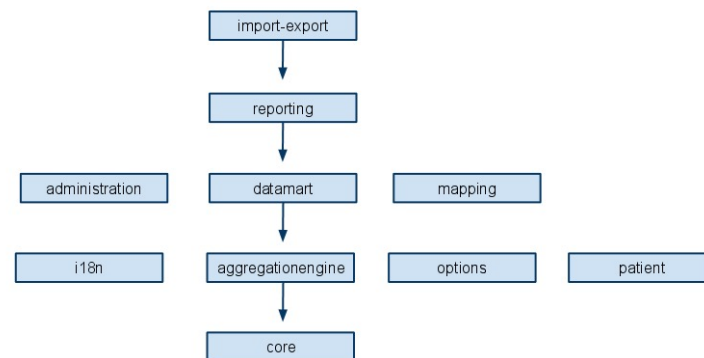
Fig. Project structure

4. 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.



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).

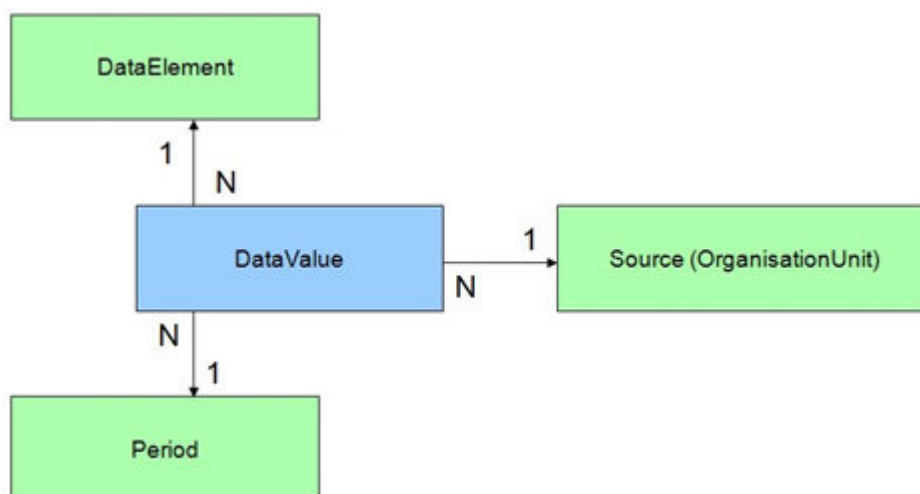


Figure 1. Data value structure

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.

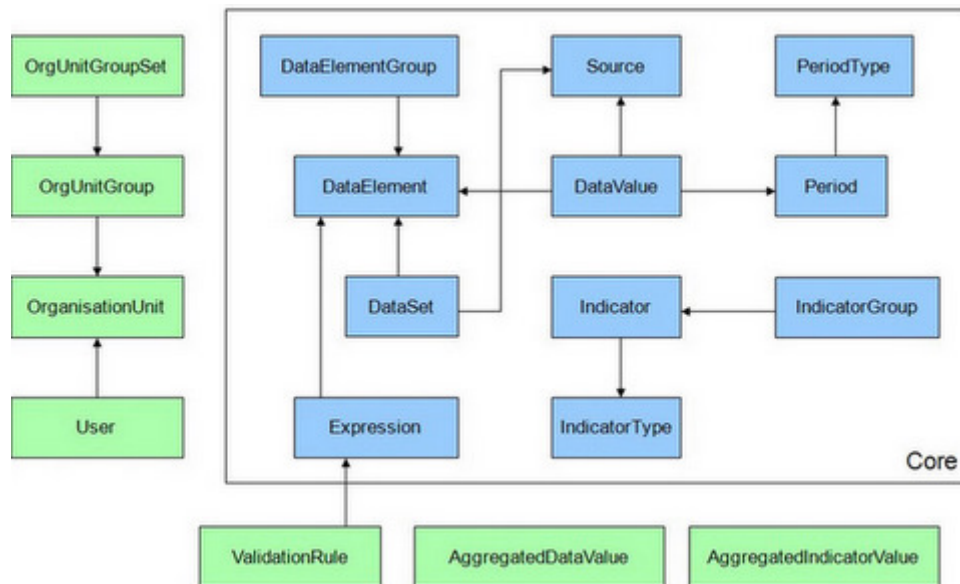


Fig. Core diagram

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.

DHIS 2 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.

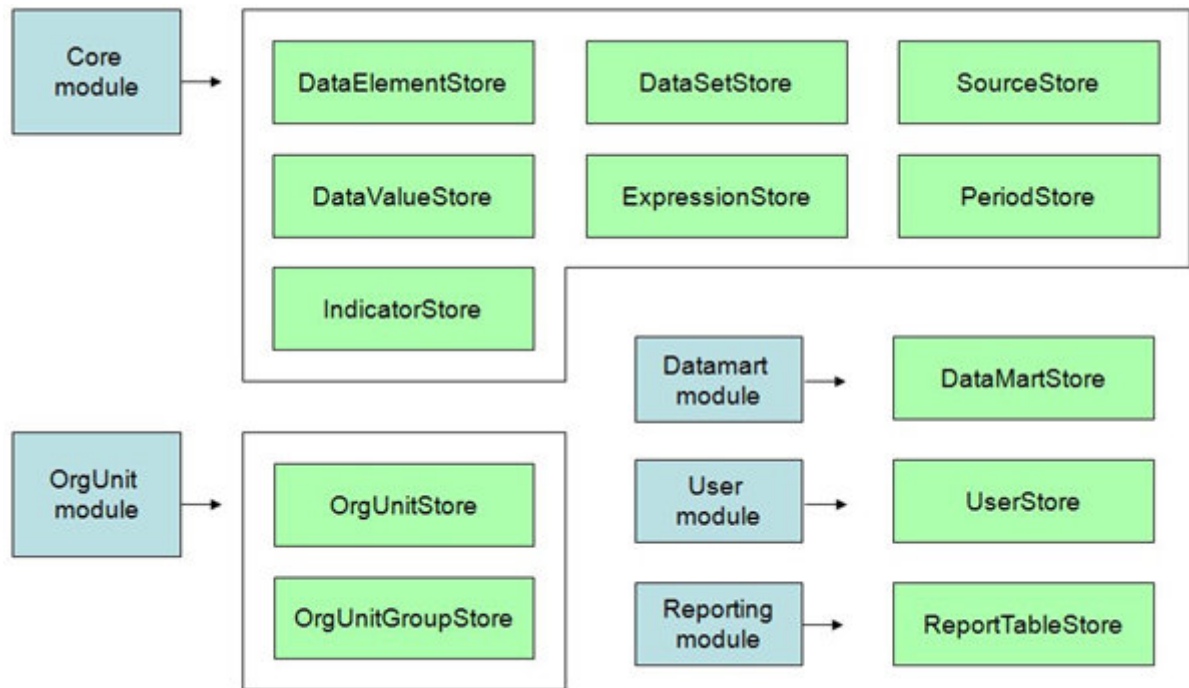


Fig. Persistence layer

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.

7.1. The JDBC Service Project

The JDBC service project contains a set of components dealing with JDBC connections and SQL statements.

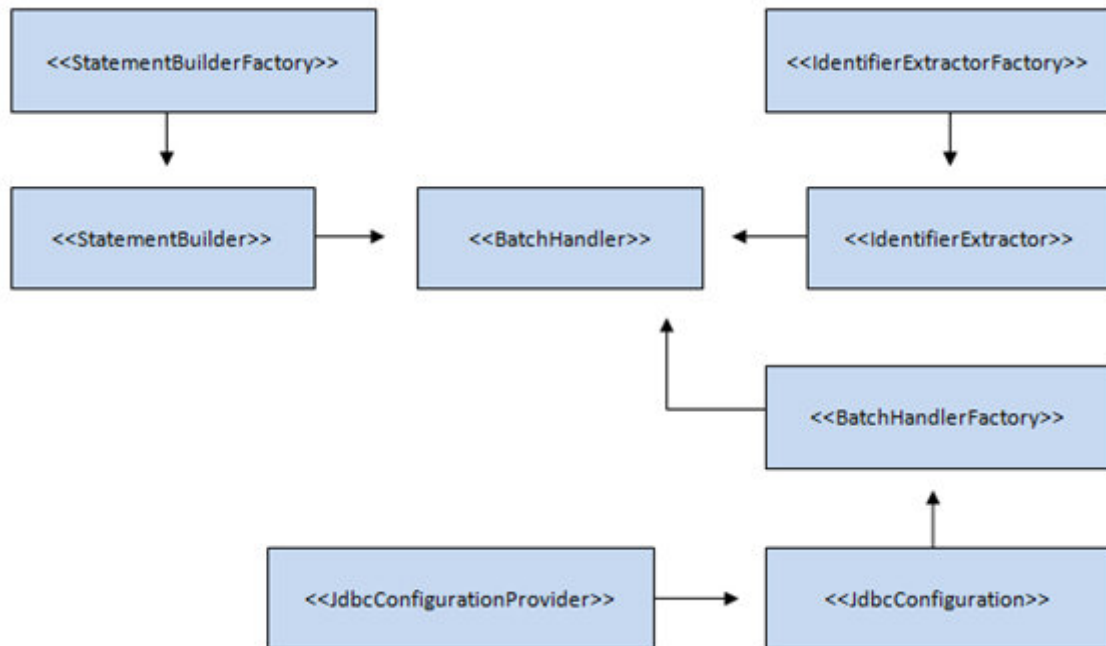


Fig. JDBC BatchHandler diagram

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.

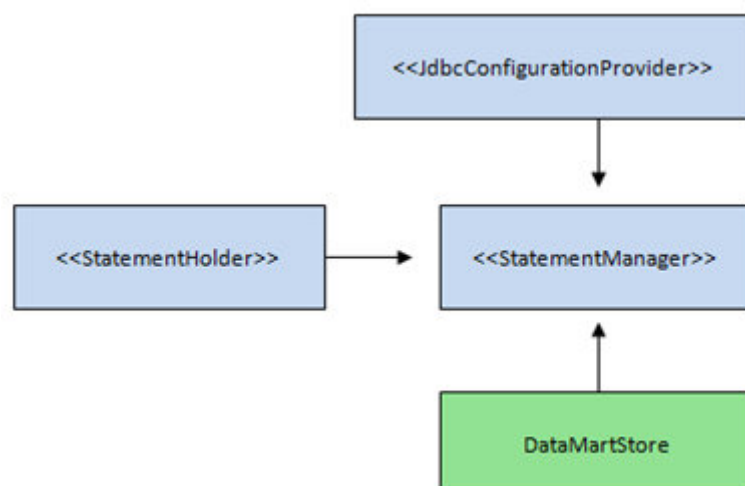


Fig. JDBC StatementManager diagram

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 *initialise()* 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*.

7.2. The Data Mart Project

The data mart component is responsible for producing aggregated data from the raw data in the time and space dimension. The aggregated data is represented by the *AggregatedDataValue* and *AggregatedIndicatorValue* objects. The *DataSetCompletenessResult* object is also included in the data mart and is discussed in the section covering the reporting project. These objects and their corresponding database tables are referred to as the *data mart*.

The following section will list the rules for aggregation in DHIS 2.

- Data is aggregated in the time and space dimension. The time dimension is represented by the *Period* object and the space dimension by the *OrganisationUnit* object, organised in a parent-child hierarchy.
- Data registered for all periods which intersects with the aggregation start and end date is included in the aggregation process. Data for periods which are not fully within the aggregation start and end date is weighed according to a factor “number of days within aggregation period / total number of days in period”.
- Data registered for all children of the aggregation *OrganisationUnit* is included in the aggregation process.
- Data registered for a data element is aggregated based on the aggregation operator and data type of the data element. The aggregation operator can be *sum* (values are summarized), *average* (values are averaged) and *count* (values are counted). The data type can be *string* (text), *int* (number), and *bool* (true or false). Data of type *string* can not be aggregated.
 - Aggregated data of type *sum – int* is presented as the summarized value.
 - Aggregated data of type *sum – bool* is presented as the number of true registrations.
 - Aggregated data of type *average – int* is presented as the averaged value.
 - Aggregated data of type *average – bool* is presented as a percentage value of true registrations in proportion to the total number of registrations.
- An indicator represents a formula based on data elements. Only data elements with aggregation operator *sum* or *average* and with data type *int* can be used in indicators. Firstly, data is aggregated for the data elements included in the indicator. Finally, the indicator formula is calculated.
- A calculated data element represents a formula based on data elements. The difference from indicator is that the formula is on the form “data element * factor”. The aggregation rules for indicator apply here as well.

the crosstabulated table gets one column for each data element. This step implies improved performance since the aggregation process can be executed against a table with a reduced number of rows compared to the raw data table.

The *DataMartService* is the central component in the data mart project and controls the aggregation process. The order of operations is:

- Existing aggregated data for the selected parameters is deleted.
- The temporary crosstabulated table is created and populated using the *CrossTabService* component.
- Data element data for the previously mentioned valid variants is exported to the data mart using the *DataElementDataMart* component.
- Indicator data is exported to the data mart using the *IndicatorDataMart* component.
- Calculated data element data is exported to the data mart using the *CalculatedDataElementDataMart* component.
- The temporary crosstabulated table is removed.

The data element tables are called “aggregateddatavalue” and “aggregatedindicatorvalue” and are used both inside DHIS 2 for e.g. report tables and by third-party reporting applications like MS Excel.

7.3. The Reporting Project

The reporting project contains components related to reporting, which will be described in the following sections.

7.3.1. 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.

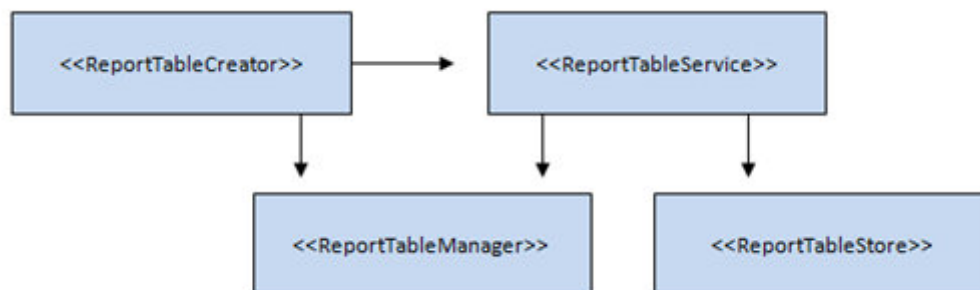


Fig. Report table diagram

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*.

7.3.2. Chart

The *Chart* object represents preferences for charts. Charts are either *period based* or *organisation unit based*. A chart has three 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.

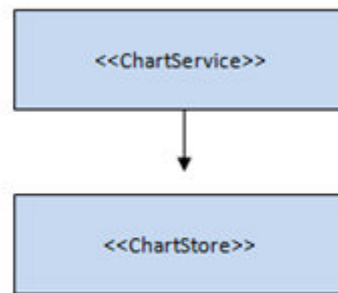


Fig. Chart diagram

7.3.3. 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.

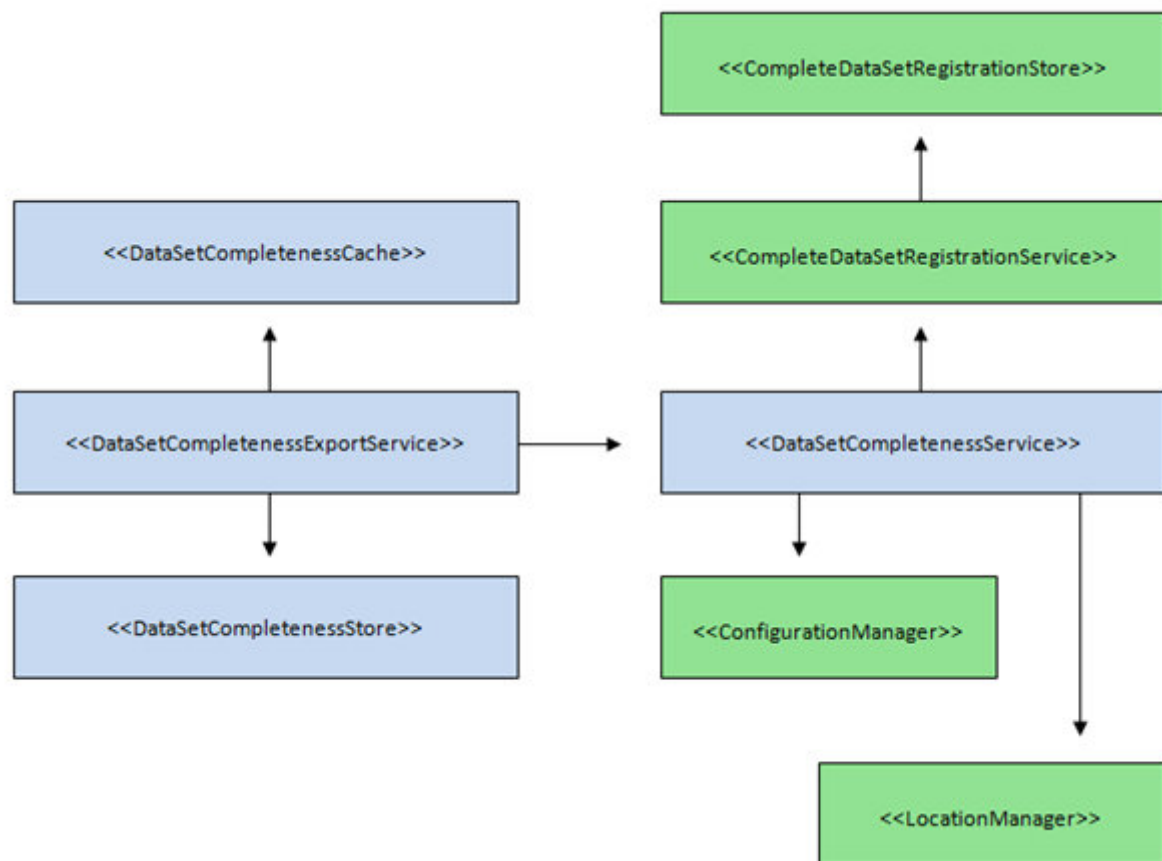


Fig. Data set completeness diagram

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 to 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 return 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 DHIS 2 for e.g. report tables and by third-party reporting applications like MS Excel. This component is retrieving data set completeness information from the *DataSetCompletenessService* and is using the *BatchHandler* interface to write such data to the database.

7.3.4. 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.

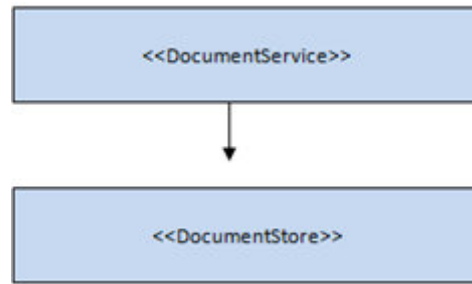


Fig. Document diagram

7.3.5. 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 responsible for creating and populating *PivotTable* objects.

7.3.6. The External Project

The *LocationManager* component is responsible for the communication between DHIS 2 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 DHIS 2. 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*.

7.4. The System Support Project

The system support project contains supportive classes that are general and can be reused throughout the system.

7.4.1. DeletionManager

The deletion manager solution is responsible for deletion of associated objects. When an object has a dependency 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 association 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 decides 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.

8. The Presentation Layer

The presentation layer of DHIS 2 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].

8.1. The Portal

DHIS 2 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.

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.

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.

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.

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.

9. Framework Stack

The following frameworks are used in the DHIS 2 application.

9.1. Application Frameworks

- Hibernate (www.hibernate.org)
- Spring (www.springframework.org)
- Struts struts.apache.org
- Velocity (www.velocity.apache.org)
- Commons (www.commonso.apache.org)
- JasperReports jasperforge.org/projects/jasperreports
- JFreeChart (www.jfree.org/jfreechart/)
- JUnit (www.junit.org)

9.2. Development Frameworks

- Maven (apache.maven.org)
- Bazaar (bazaar-vcs.org)

10. 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. R and DHIS 2 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. 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 [Section A.4, “Mapping with R and PostgreSQL”](#)

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, let's 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 it 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.3. Using R with MyDatamart

MyDatamart provides a useful interface to the DHIS2 database by making a local copy of the database available on a user's desktop. This means that the user does not need direct access to the database and the data can be worked with offline on the user's local machine. In this example, we will have used the [demo database](#). Data was downloaded at the district level for Jan 2011-Dec 2011. Consult the MyDatamart section in this manual for more detailed information.

First, let's 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 retrieve all indicators
#at OU2 level
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 let's 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 a 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), let's further subset the data so that we only have these.

```
#Subset the data for ANC
ANC<-Inds[grepl("ANC (1|2|3) Coverage",as.factor(Inds$indshort)),]
```

We just used R's **grepl** 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 let's fix this before we try and create a plot of the data.

```
#Let's reorder the months
MonthOrder<-c('Jan', 'Feb', 'Mar', 'Apr',
'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
ANC$month<-factor(ANC$month,levels=MonthOrder)
```

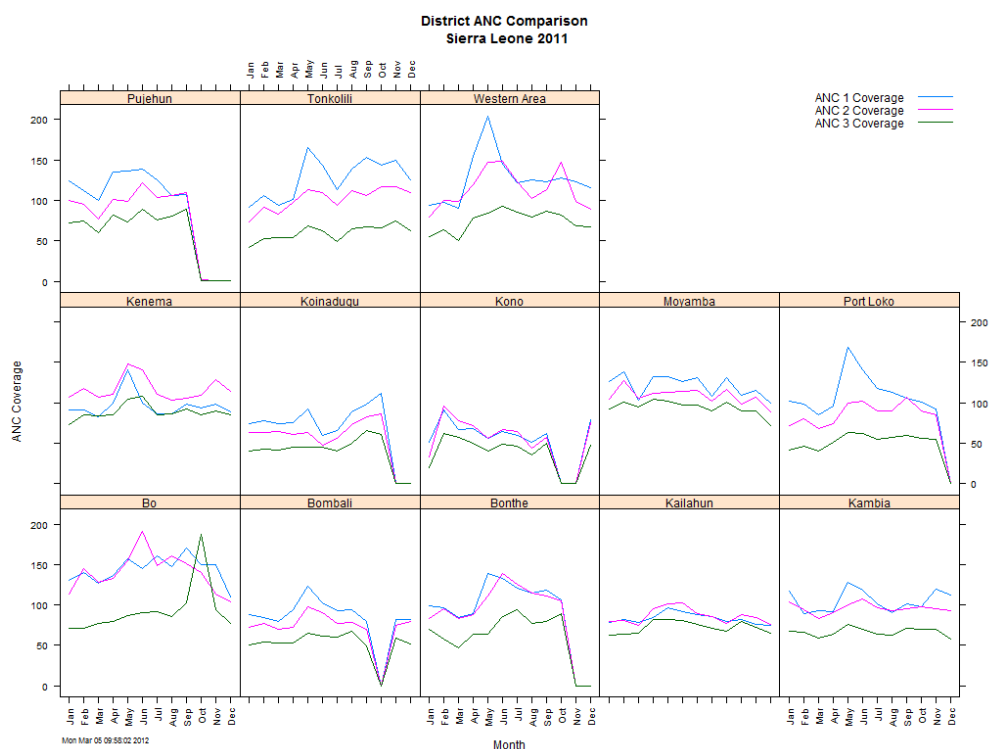
Next, we need to actually calculate the indicator value from the numerator, factor and denominator.

```
#Calculate the indicator value
ANC$value<-ANC$numxfactor/ANC$denominatorvalue
```

Finally, let's 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.4. 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]+\.[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 indicators with the coordinates data frame
  myDF<-merge(myDF,coords)
  #We need to cast the new data frame 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)

```

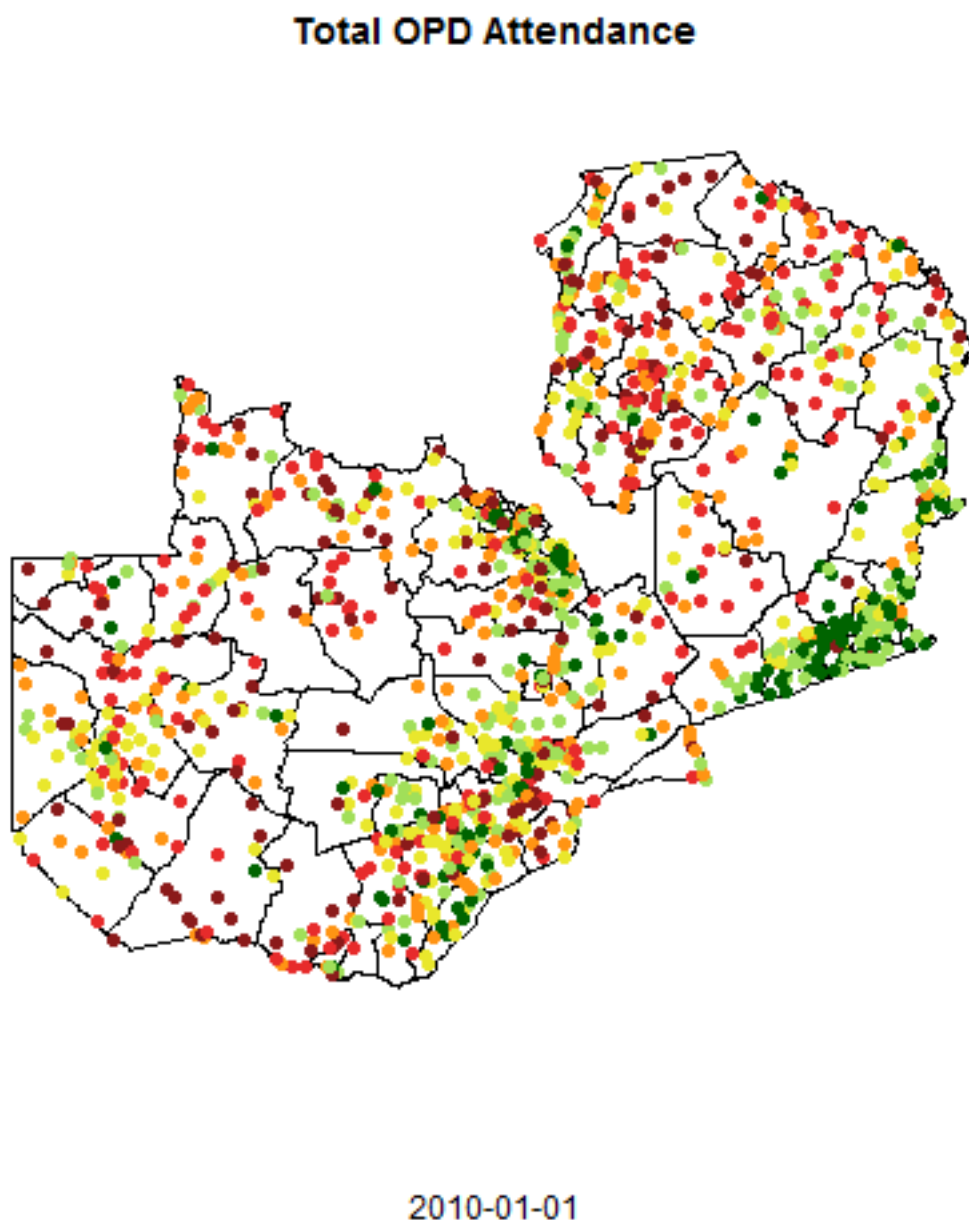


```
#Finally, return the plot from the function  
return(myPlot) }
```

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.



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.5. 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')"
```

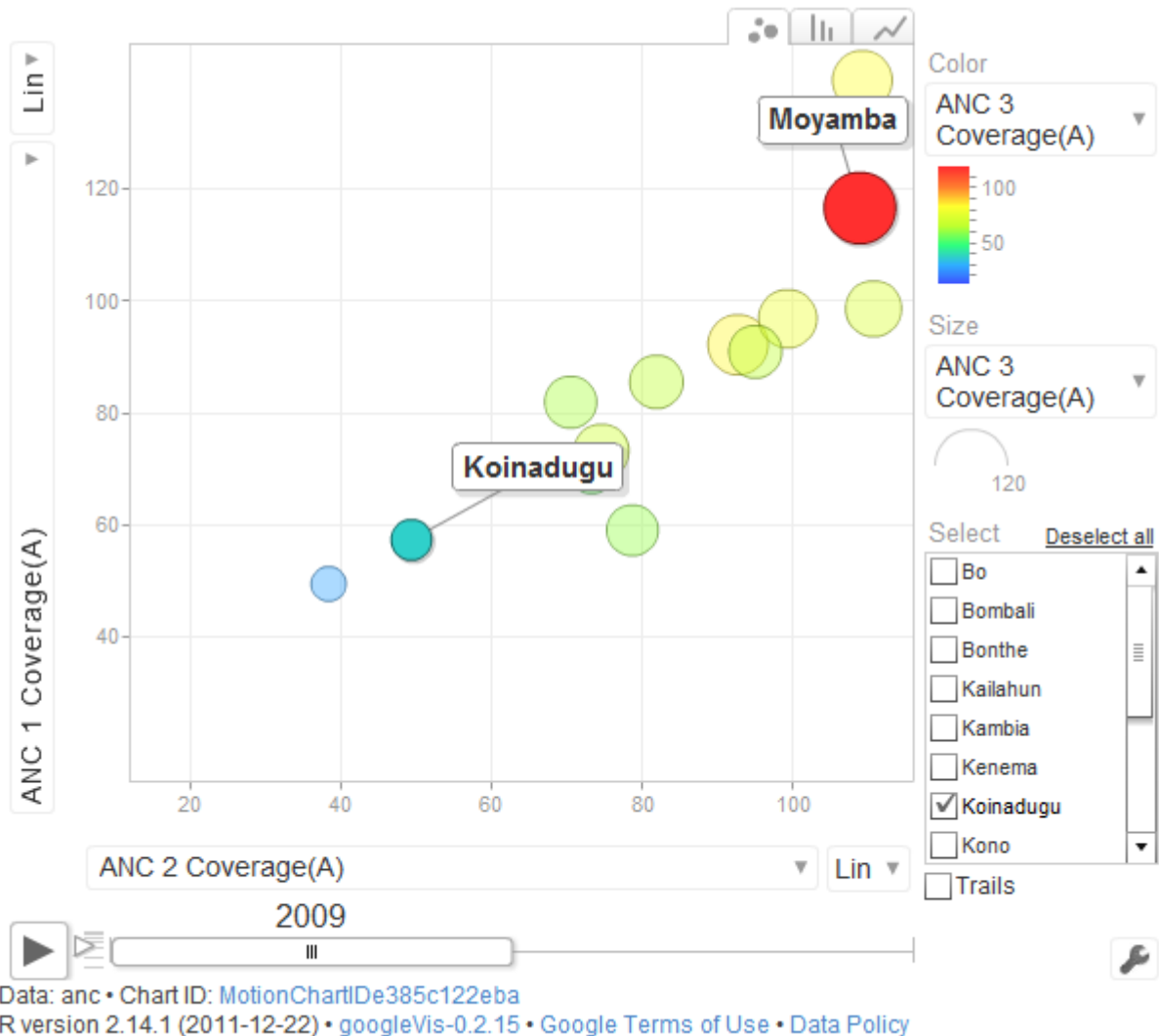
```

#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.6. 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.7. 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://play.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://play.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.