IATI API Conventions / Standards

This is a 0.1 document produced as the result of a meeting of the IATI community in November 2012 and revised following conversations in January 2013.

A slide deck summarising key issues discussed on the 15th January is here.
There are also some notes from the meeting

RFC Draft: International Aid Transparency Initiative APIs

This document uses the terms MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY according to RFC2119 (http://www.ietf.org/rfc/rfc2119.txt).

What is an IATI API?

An IATI Application Programming Interface (API) is a programming interface to give access to all, or a subset of, data published according to the International Aid Transparency Initiative standard (www.iatistandard.org).

Any APIs MAY serve a range of purposes and offer a range of different features, including simple access to data, search over data fields, alternative serialisations of data, geographic search, and returning aggregate figures from data.

Terminology

The following terminology and abbreviations are used in this document:

- **Publishers -** used to refer to any entity publishing a file using the IATI standard. Publishers could include donors, implementing agencies and third-parties providing additional data about an aid activity. Publishers will have produced at least one organisation or activity file.

- **IATI Standard** - includes both the Organisational File and Activity File standards

- **Organisation File -** is an XML file prepared to the IATI **Organisation Standard** (http://iatistandard.org/organisation-standard).

- **Activity File** - is an XML file prepared to the IATI **Activity Standard** (http://iatistandard.org/activities-standard)

- **IATI Activity** - is a single aid activity contained within the <iati-activity></iati-activity> elements of an Activity File XML document.

- **IATI Standard Version** - refers to versions of the Organisation and Activity Standards. These are synchronised, so that an increase in version number of one is matched by an increase in the version number of the other. A **schema** is published for each version of the Organisation and Activity standards

- **IATI Codelists** - refers to the codelists provided alongside the IATI Standard.

- **IATI Registry** - is a data catalogue that records meta-data on IATI Activity Files, IATI Organisation Files, and meta-data on Publishers. It is accessible through the **Registry API**

- **Registry API** - provides programmatic access to the data catalogue of available IATI Activity Files and IATI Organisation Files.

Why do we need an API standard?

The International Aid Transparency Initiative requires that organisations publish XML files (a) about their organisations; and (b) about aid activities that they are involved in. The segmentation policy of IATI (http://iatistandard.org/guides/data-considerations/segmenting-your-data) means that donors should publish one IATI Activity file per region or country they work in, and that an activity should only ever be included in one file. This means that users cannot be sure they have found all the activities for a given country, or a given sector, without searching through all available IATI Activity files. It also means that there are considerable technical barriers for anyone wishing to use IATI data, requiring them to download, parse and manipulate XML data.

Many uses of IATI data therefore involve placing the data into some form of data store or database, and their retrieving it from there, without primary regard to the separate files it was distributed in. Many such stores of IATI data will provide API access to their contents, and to analytic functions over the data. If each available service doing so takes a different approach to providing access to that data, and serialising the results of queries, then users still face a high barrier when encountering any new platform, and tools built on top of these platforms will need to be customised for each data store they use. An API Standard will increase the interoperability of tools for working with IATI data, and will lower the barriers for anyone seeking to use the data.

An API standard can also help developers to address some of the complexity of working with IATI data by recording common patterns and logic for dealing with difficult issues such as versioning, double-counting, JSON and CSV serialisation of XML data, and so-on.

Balancing standardisation and flexibility

A structured set of data can be queried in many different ways. The expressive power (and complexity) of languages like SQL and xQuery demonstrates just how many different ways it might be possible to query and work with even a small set of IATI data. The goal of an API Standard is not to replicate these rich query languages, but is to provide some basic common conventions and patterns for handling and accessing IATI data.

This means that:

- It **should** allow users to switch between APIs providing comparable features without making substantial changes to their code or working practice

- It **should not** prevent API from implementing advanced and specialist features, or serving particular niche needs

Some user needs will still be best served by processing IATI data directly, rather than using a data store and API.

Design principles

A number of design principles guide the development of an IATI API.

1) RESTFul API

APIs should follow a RESTful pattern, using URLs for key services, and query string parameters for any filters and settings to be applied to responses.

2) Fidelity to the IATI standard

The IATI Schema provides clear definitions of a wide range of elements.

Names used in any API should be based closely upon the names in the IATI Standard schemas. The schemas and code lists provide the authoritative definition of terms. The use of nested elements in the IATI XML schema is semantically important.
The rendering of element names and structures in different serialisations (e.g. CSV, JSON) should use predictable rule-based transformations of the element names and structures from the schema.

3) Query by codes, rather than names

IATI has been designed (a) to support multilingual data; (b) to enable users to review individual XML files without reference to code list look-up tables. This means there are places where both a code or reference, and a text description can be given against an element. The text descriptions of code list entries can vary across language, but the codes will always stay the same. For this reason, the default behaviour should be to use codes in queries rather than the text of elements.

4) Accountability and provenance

Most APIs will perform some transformation on the raw IATI XML data they consume. It is important for users to be able to discover (a) what IATI data any public API covers; (b) what processes have been applied to data to clean, transform or augment it.

Importing and transforming data

API platforms will generally import and cache IATI XML data before providing access to it

through an API. API platforms MAY choose to pre-process or transform the data in some way, either at import or query time. API platforms should keep a record of any transformations made, and SHOULD make this accessible in a human readable form. They MAY make it accessible in a machine readable form.

No convention for expressing this provenance information has yet been agreed.

A number of common transformations are address in a following section.

Endpoints

APIs MAY choose to provide any of the following endpoints. Any additional endpoints SHOULD be added according to the convention below. In the case of any ambiguity over what to name a new endpoint please consult with the IATI Technical mailing list.

The endpoints follow the pattern of using the singular name of an element for retrieval of a single record, and the english plural for endpoints that will list records. Singular endpoints accept a single value (the record identifier) at the end of the URL. List endpoints accept multiple parameters (specified below).

> The use of separate access/ and aggregate/ endpoints allows a provider to easily use different back-end systems where this is required to meet the computationally distinct requirements of returning records in response to a query, and providing totals by calculation across records.

access/ - for all queries that return lists, collections or individual records

> **activity/** - for returning a list of activities.
>
> **activity/{iati-identifier}** - for returning a single activity. APIs SHOULD NOT expose any identifier other than their iati-identifier.
>
> **transaction/** - for returning a list of transactions.
>
> **transaction/{transaction-identifier}** - for returning a single transaction.
>
> > *Note: {transaction-identifier} is not specified by the standard, and it is possible in data from multiple sources that transaction/@ref will not be a unique value. For this reason, APIs providing direct access to individual transactions will need to create their own transaction-identifier.*
>
> **organisations/** - for returning a list of organisations based on organisational files

organisation/{reporting-org/@ref} **-** for returning a single organisations details based on organisation files

A number of entities occur across activities, and have no single unique representation within IATI Activity or Organisation files. This includes sectors, countries, organisations and classifications applied to activities. Endpoints may be provided which return a general description of these entities based upon code lists and/or upon information derived from the available data. Examples include:

participating-org/ - providing a list of know participating organisations

(To get just funding organisations use /access/participating-orgs/?role=Funding)

participating-org/{@ref} - information on a specific participating organisation

sector/{@vocabulary} - providing a list of sectors. {@vocabulary} is optional.

sector/{@vocabulary}/{@code} - providing details of a specific sector

recipient-country/{@code} - providing details of a specific country

recipient-region/{@code} - providing details of a specific region

aggregate/ - the aggregate endpoint should be used for all aggregations across activities or organisations.

(Note: some APIs MAY also choose to provide per-activity aggregation: for example, to add up the total value of transactions by type for an activity, and to add this to the data returned against that activity. This type of activity-level aggregation is exposed under the access/ endpoint.)

Under aggregate/ the following common endpoints may be provided:

activity/ - for total numbers of activities in an area

transaction/ - for aggregating transactions

budget/ - for aggregating budget values

planned-disbursement/ - for aggregating planned disbursements

organisation/

provenance/ - this endpoint could be used for machine readable provenance data. Not detailed use case or specification is worked out yet.

about/ - this endpoint should return a human and machine readable description of the API, including information about the creator, the features the API implements, and any special considerations for users of the data. No detailed specification for this is worked out yet.

Versioning

An endpoint without version numbers should point to the most recent version of the endpoint. Version numbers may be prefixed to address a particular version of an API. For example, if the current version is 1.2 then:

/1.1/access/ talks to version 1.1
/1.2/access/ talks to version 1.2
/access/ talks to version 1.2

Please detail in /about/ and in API responses which version of the IATI standard you are using in rendering output.

Query parameters

List endpoints MAY accept parameters used to filter their output. APIs MUST accept parameters as part of the querystring in a GET request. APIs MAY accept parameters in a POST request.

Users may want to query against top-level elements of an IATI Activity or Organisation file, or against nested elements.

Parameter naming

The default naming pattern for parameters should be:

> {parent-element-name}_{element-name}.{attribute-name}

Where the parent is iati-activity or iati-organisation, then {parent-element-name}_ may be omitted.

When .{attribute-name} is omitted, the API should use a default behaviour based on the following rules:

- For elements where the schema allows @code or @ref values, then search on the value of @code or @ref
- For elements without @code or @ref specified, then search on the value of the element

Examples:

- ?sector=60061 searches on //sector[@vocabulary='DAC']/@code
- ?sector.text =Debt searches on //sector[@vocabulary='DAC']/text()
- ?participating-org=GB-1-123 searches on //participating-org/@ref
- ?participating-org.text=Oxfam searches on //participating-org/text()
- ?iati-identifier=GB-1-123 searches on //iati-identifier/text()
- ?location_name=Oxford searches on //location/name/text()
- ?transaction_value=1000 searches on //transaction/value/text()

Where a default behaviour is provided it MUST be possible to also use the canonical parameter name. For example, ?sector.code and ?sector should both be valid to search //sector[@vocabulary='DAC']/@code

It MUST be possible to search on the element value with .text.

Logic

When multiple distinct parameters are provided the default behaviour MUST be to combine these on the basis of 'AND'.

For example, /access/activities/?recipient-country=AF&reporting-org=GB-1 would lead to the query:

//iati-activity[recipient-country/@code='AF' **and** reporting-org/@ref='GB-1']

Some parameters may accept multiple values. These can be separated with '|' to indicate that the values should be combined with an 'OR' operation. To perform an AND operation the parameters should be repeated.

For example: /access/activities/?recipient-country=KE|UG would lead to the query:

//iati-activity[recipient-country/@code='KE' **or** recipient-country/@code='UG']

whereas /access/activities?receipient-country=KE&recipient-country=UG would lead to the query:

//iati-activity[recipient-country/@code='KE' **and** recipient-country/@code='UG']

APIs MAY choose to allow wildcards in searches

- * = any value
- % = a single character

Query types

By default '=' means exact match. (The additional ?q and ?query-fields parameters detailed below can be used to provide a default fuzzy match search over fields)

A query can be modified using the following suffixes on the parameter. These are preceded by a

double underscore. The following SHOULD be provided by all APIs for free text, date or numerical parameters:

- __gt - greater than (numerical and date parameters)
- __lt - less than (numerical and date parameters)
- __contains - contains (free text parameters)
- __icontains - case insensitive contains (free text parameters)
- __iexact - case insensitive exact match (free text parameters)

The expected behaviour for a __gt or __lt search over a date field is that data comparison will be used. Over numeric fields the expected behaviour is a numerical sort.

An API MAY choose to implement other modifiers. We recommend using the [django lookup types] as a starting point, and consulting with the IATI Technical list if implementing any lookup types not included on that list.

Additional parameters

The following additional parameters MAY be provided:

- **?q -** for running a case insensitive contains query over free text. The default behaviour would be to search title, description and sector text. For example ?q=debt would search:
  //iati-activity[contains(lower-case(title),lower-case($query)) **or** contains(lower-case(description),lower-case($query)) **or** contains(lower-case(sector),lower-case($query))]

- **?query-fields** - to modify a ?q query and specify the fields that a search should be run over. For example, specifying ?q=goma&query-fields=title|location_name would search:
  //iati-activity[contains(lower-case(title),lower-case($query)) **or** contains(lower-case(location/name),lower-case($query))]

The following parameters SHOULD be used to control output.

- **?format=** - for choosing whether to retrieve JSON, XML, CSV and so-on
  - Valid options include xml, json, csv, rdf and html
  - If csv is selected, a default csv option SHOULD be supplied, but in most cases this would be combined with a &response-profile parameter to specify an approach to flattening data to be used.

- **?lang=** - APIs MAY offer the user a choice of what language to retrieve text and codelist fields in where multiple languages are available. Languages should be specified using ISO 639 codes.

- **?limit=** - the number of records to return

- **?start=** - the offset to use in fetching records. E.g. if ?limit=10 then &start=11 fetches the second page of results.

The following parameters MAY be offered to control output

- **?fields=** - APIs MAY offer the user the chance to specify a comma separated list of elements to include in the response, using the same naming convention as for querying. The following keywords could also be made available:
    - All - return all fields

- **?fields-exclude=** to exclude particular elements from the response

- **?response-profile=** to specify how much detail should be included in the returned data. The exact treatment of this parameter may vary between APIs, but as a outline suggestion we might have:
    - **codes** - omit all but essential text descriptions and only provide codelist codes (e.g. for sector)
    - **summary** - provide only top-level fields about an activity, excluding transactions etc.
    - **extended** - provide additional information, such as annotations on organisations, sector codes etc.

- **?sort-by=** specifying a comma separated list of fields to sort by

- **?sort-order** = ASC|DESC - specifying the order to sort in

- **?split-by** = used in constructing CSV output

Return formats

APIs may provide a range of different formats in which data is returned.

Response headers

The response should be wrapped within an iati-activities or iati-organisations block, with attributes for:

- **version** - the version of the IATI standard used in the output
- **generated-datetime** - when the response was generated/cached
- **default-currency** - the default currency used (optional)

The following response headers may be provided, wrapped within a query block (and serialised using the same approach as specified for different formats below. In the case of CSV, this could be written to a header or footer. These are defined according to their parameter definitions above and would have the parameter passed, or default value, set unless otherwise specified:

- **total-count -** the total number of results available
- **warnings** - and specific considerations as user should pay attention to. Particularly important for aggregate queries, where double-counting or mixed currency warnings may be appropriate.
- **limit**

- **start**
- **lang**
- **response-profile**
- **sort-by**
- **sort-order**
- any other parameter

For example, the query: access/activities?recipient-country=GH&limit=100&start=101 would have the header:

```
<iati-activities version="1.01" generated-datetime='2011-09-26T13:40:52.98' default-currency='GBP'>
    <query>
        <total-count>521</total-count>
        <limit>100</limit>
        <start>101</start>
        <lang>en</lang>
        <response-profile>default</response-profile>
        <sort-by>iati-identifier</sort-by>
        <sort-order>ASC</sort-order>
        <recipient-country>GH</recipient-country>
    </query>
    …
    results
    ...
</iati-activities>
```

XML serialisation

XML serialisation should follow the IATI standard as closely as possible. With the exception of allowed additions, augmented and additional data should be provided in application-specific namespaces.

CSV Serialisation

There is no simple way to flatten XML as CSV. However, we can assume a number of defaults unless over-ridden using a response-profile or group-by parameter.

By default:

- Queries to an activity endpoint should return one row per activity;
- Queries to a transaction endpoint should return one row per transaction. Transaction responses should always include an additional column for the related activity iati-identifier, and may choose to also include other activity meta-data;
- Queries to an organisation endpoint should return one row per organisation;

The default approach for column names should follow the naming of parameters (e.g.

```
title,recipient-country,recipient-country.text,description
Project A,AF,Afghanistan,A project in Afghanistan
```

Where a field has multiple values, these should be concatenated into a cell using ';'. For example, if two recipient countries are given:

```
title,recipient-country,recipient-country.text
Project B,KE;UG,Kenya;Uganda
```

Where an element has attributes, these should precede the text() of the element in the CSV structure. For example, with the allowed additional data for total-commitments etc. the output should be:

```
total-commitment.currency,total-commitment.value-date,total-commitment
GBP,2010-10-01,230000
```

An API optimising CSV for human readability MAY choose to replace '-' and '.' in output with spaces, and to capitalise column headings.

Alternative CSV output can be provided using **either:**

- **?response-profile=** to select some pre-determined CSV serialisation approach. At present no library of approaches exists.
- **?split-by=** to provide one or more comma separated elements which should be used to determine duplication of rows.

For example /access/activities/?format=csv&split-by=sector would produce a file which repeats each iati-activity row once for each sector it contains. An example output:

```
iati-identifier,title,sector,sector.percentage,sector.text,recipient-country
GB-1-123456,Project A,31166,Agricultural extension,50,KE;GH
GB-1-123456,Project A,11330,Vocational training,50,KE;GH
GB-1-876543,Project B,11420,Higher education,100,UG
```

Using two or more ?group-by options would increase the row duplication further. For example: /access/activities/?format=csv&split-by=sector,recipient-country would generate:

```
iati-identifier,title,sector,sector.percentage,sector.text,recipient-country
GB-1-123456,Project A,31166,Agricultural extension,50,KE
GB-1-123456,Project A,11330,Vocational training,50,KE
GB-1-123456,Project A,31166,Agricultural extension,50,GH
GB-1-123456,Project A,11330,Vocational training,50,GH
GB-1-876543,Project B,11420,Higher education,100,UG
```

JSON serialisation

JSON SHOULD map to the structure of the XML as closely as possible, with element names and nesting identical to the XML Schema, even when providing abbreviated data.

Note, this requires that property names are always quoted, to avoid validation errors because of the presence of hyphens in element names. Quoting property names is good practice in JSON in any case.

Where an element has attributes, return an object, using the following rules to set the attributes:

- xml:lang becomes ‘lang’
- If the element schema shows the element type is ‘currencyType’ place the value of text() in an attribute named ‘value’, otherwise place it in an attribute named ‘text’.
- Name all other json attributes after the xml attribute

For example:

```
<iati-activity>
   <iati-identifier>21020-3DFMYAN</iati-identifier>
   <reporting-org ref="21020" type="21">International HIV/AIDS Alliance</reporting-org>
   <title xml:lang='en'>Three Diseases Fund for Myanmar</title>
   <description type="1">The three dieseases fund aims to contribute to achieving the UN Millennium Development goals in Myanmar by reducuing the burden of communicable disease morbidity/mortality. Activities will be undertaken to reduce transmission and enhance provision of treatment and care for HIV and AIDS,turberculosis (TB) and malaria</description>
   <activity-date type="start-actual" iso-date="2007-05-15">2007-05-15</activity-date>
   <transaction>
      <value value-date="2011-09-11">2096562</value>
      <transaction-type code="C">Commitment</transaction-type>
      <transaction-date iso-date="2011-09-11">Total Project Budget</transaction-date>
   </transaction>
</iati-activity>
```

would become:

```
{
  "iati-activity": {
    "iati-identifier": "21020-3DFMYAN",
    "reporting-org": {
      "ref": "21020",
      "type": "21",
      "text": "International HIV/AIDS Alliance"
    },
    "title": {
      "lang": "en",
      "text": "Three Diseases Fund for Myanmar"
    },
    "description": {
      "type": "1",
      "text": "The three dieseases fund aims to contribute to achieving the UN Millennium Development goals in Myanmar by reducuing the burden of communicable disease morbidity/mortality. Activities will be undertaken to reduce transmission and
```

```
enhance provision of treatment and care for HIV and AIDS,turberculosis (TB) and malaria"
    },
    "activity-date": {
      "type": "start-actual",
      "iso-date": "2007-05-15",
      "text": "2007-05-15"
    },
    "transaction": {
      "value": {
        "value-date": "2011-09-11",
        "value": "2096562"
      },
      "transaction-type": {
        "code": "C",
        "text": "Commitment"
      },
      "transaction-date": {
        "iso-date": "2011-09-11",
        "text": "Total Project Budget"
      }
    }
  }
}
```


If an API is only providing a single language (either as a default behaviour, or by request), and the only attribute of an element is xml:lang, then it may choose to return the value as a string, rather than an object.

For example:

```
"title":  "Three Diseases Fund for Myanmar"
```

rather than:

```
"title": {
    "lang": "en",
    "text": "Three Diseases Fund for Myanmar"
 },
```

RDF serialisation

To be agreed

HTML serialisation

APIs may choose to provide a HTML rendering of responses. No standard is proposed for this.

Common additions

APIs may choose to add some convenience aggregations to their output for activities, such as aggregating transaction values, or providing a start-date and end-date based on the planned and actual start and end-dates.

Aggregation elements should be lowercase, and prefixed with ‘total-’. They should be generated using the following rules:

For transactions:

Name the element after the english language code description for transaction/transaction-type/@code

For example, the aggregate of all Commitments for an activity would be placed in an element named **total-commitment** as follows:

<iati-activity>
    <iati-identifier>GB-1-123456</iati-identifier>
    ….
    <total-commitment currency=’GBP’ value-date=’2010-10-7’>100000</total-commitment>
</iati-activity>

Note that the element otherwise inherits the properties of transaction/value.

For budgets

Name the element total-budget

For example:
<iati-activity>
    <iati-identifier>GB-1-123456</iati-identifier>
    ….
    <total-budget currency=’GBP’ period-start=’2010-01-01’
period-end=’2012-01-01’>100000</total-commitment>
</iati-activity>

It is possible to use total-budget multiple times to provide yearly budgets for an activity that provides quarterly budgets and so-on.

Start and end-dates

APIs may wish to accept queries against the start-date and end-date of an activity to escape the complex logic involved in querying for activity-date by types.

They may add a ‘start-date’ and ‘end-date’ element directly to the activity based on the convention that:

- start-date = activity-date[@type=’start-actual’] or, if this is missing, then activity-date[@type=’start-planned’]
- end-date = activity-date[@type=’end-actual’] or, if this is missing, then activity-date[@type=’end-planned’]

The @type attribute on these elements should be set, so that a user can clearly determine whether the start-date and end-date are actual or planned dates. For example:

```
<activity-date type="start-planned">2006-04-01Z</activity-date>
<activity-date type="start-actual">2007-01-01Z</activity-date>
<activity-date type="end-planned">2009-12-31Z</activity-date>
```

Could lead to the additional elements being added to an activity:

```
<start-date type="start-actual">2007-01-01Z</start-date>
<end-date type="end-planned">2009-12-31Z</end-date>
```

Some APIs may also wish to add additional date parameters, particular to support aggregate operations. These include:

- **start-year / end-year**
- **start-quarter / end-quarter** - Note, should use calendar year quarters
- **start-month / end-month**
- **start-day / end-day**

Hierachical codelists

For code lists like the DAC which have an implied hierarchy, APIs may choose to add the higher level sector category to their data store using a distinct vocabulary. As a convention, this vocabulary should be named {VOCAB}-PARENT. For example, DAC-PARENT

E.g. a file containing:
<sector vocab='DAC' code='60061'>Debt for development swap</sector>
could have the additional line added:
<sector vocab='DAC-PARENT' code='600'>Activity relating to debt</sector>

It then becomes possible to query using:
?sector=600&sector.vocabulary=DAC-PARENT

Other additions?

Please suggest approaches and logic for other commonly required additions on the IATI Technical mailing list.

Aggregation

The aggregation endpoint exists to support aggregation across activities or organisations according to some set of query parameters.

Each aggregation query will include the following parameters:

- **group-by** - one or more elements or attributes that will be used as dimensions in the return aggregation
- **aggregate-function -** count|sum|max|min|mean etc. APIs can choose which functions to implement, but should generally implement count and sum.
- **aggregate-element -** the element to be the target of aggregation for sum, max, min, mean etc. queries.
    - Defaults may be set for different end-points. For example, default to /transaction/value/text() for the /aggregate/transactions/ endpoint.

Aggregation queries can also be subject to all the same filter parameters as other queries. They may choose to implement the sorting and paging parameters of other queries. An additional keyword for sort-by in aggregate queries will be ‘aggregate-element’. E.g. get a list of countries in order of the one with most projects.

Values are returned in rows each with a number of dimensions and a measure.

A simple example:

/aggregate/activities/?group-by=recipient-country&aggregate-function=count&sort-by=aggregate-element&sort-order=DESC

would generate:

```
<row>
        <recipient-country>UG</recipient-country>
        <count>10</count>
</row>
<row>
        <recipient-country>KE</recipient-country>
        <count>6</count>
</row>
```

Where the element to be aggregated has multiple types (this primarily/solely applies to transactions) then the return should include these, unless otherwise specified by a filter. For example:

/aggregate/transactions/?group-by=recipient-country&aggregate-function=sum&aggregate-element=transaction_value&transaction_transaction-type=C|E

would return:

```
<row>
        <recipient-country>UG</recipient-country>
        <total-commitment>10634</total-commitment>
        <total-expenditure>10234</total-expenditure>
```

```
</row>
<row>
        <recipient-country>KE</recipient-country>
        <total-commitment>631234</total-commitment>
        <total-expenditure>212334</total-expenditure>
</row>
```

Note, that in the case of **sum** transactions we are applying the naming rules for **common additions** from above. This assists in intuitive naming of results for end users.

This handling of multiple types is a special rule, and **further testing is required** to identify if this pattern meets the guiding principles for our API design.

Where multiple group-by elements are provided, then these should all be evaluated as dimensions. For example, the query:

/aggregate/transaction/?groupby=receiver-org,start-year,start-quarter&aggregate-function=sum&transaction_transaction-type=C

would return:

```
<row>
        <receiver-org>GB-COH-1213512</receiver-org>
        <start-year>2010</start-year>
        <start-quarter>2</start-quarter>
        <total-commitment>500</total-commitment>
</row>
<row>
        <receiver-org>GB-COH-1213512</receiver-org>
        <start-year>2010</start-year>
        <start-quarter>3</start-quarter>
        <total-commitment>7000</total-commitment>
</row>
...
<row>
        <receiver-org>US-DEL-12356</receiver-org>
        <start-year>2010</start-year>
        <start-quarter>2</start-quarter>
        <total-commitment>50000</total-commitment>
</row>
etc.
```

Aggregation notes of caution:

- A budget/@type='revised' should always replace a budget/@type='original' for the same period to avoid double-counting.

- Care should be taken to avoid double-counting the same information reporting by multiple donors

- Care should be taken to avoid double counting for activities classified against multiple recipient countries or sectors. It may be appropriate to apportion transactions according to country and sector percentages.

- Care should be taken to avoid mixing transactions, budgets and other elements which have different currency values.

Special considerations

Code lists
The codelists should be readable.

Rolling information up to the activity level:
In the case of hierarchical data, it may be appropriate to add total-commitment elements etc. from hierarchy=2 activities to their parent activities.

Apportioning country allocations
Aggregation applications computing data cubes may wish to apportion transactions to countries or sectors on the basis of the percentages given against recipient-country and recipient-sector.

Double counting

Managing different standard versions

Handling languages

Pre vs post processing

RESTful URLs

Some APIs may wish to provide additional RESTful URLs

Processing

Transaction Reference
Transaction elements MAY have a @ref attribute used to "describe reference to this transaction in another system".

Recommendations on steps that should be taken during importing to handle bad data or to tidy up data.

- Codelist expansion / trust official code lists
- Currency conversion
- Location expansion (e.g. regional containment?)
- Date expansion

Recommend that API platforms using non-XML data storage to record the relationship between any XML entity and given database fields. This would support the specification of CSV serialisations in terms of the xpath expression for a given field.

Additional data in responses

Processes - can we develop a peer-review process. Submit anything to peer-review from the community.

Other considerations

http://www.apievangelist.com/buildingblocks/

Background information

Workshop Plan

27th November 2012 - DFID, 1 Palace Street, London, SW1E 5HE

Goals:

- To identify common patterns and principles for API access onto IATI data;
- To draft an 'API Contract' that specifies what consuming applications can expect from an IATI API. Identifying 'required', 'recommended' and 'optional' aspects of an API;
- Agreeing draft timetables for API harmonisation;

Participants

- John Adams - DFID
- Tim Davies - AidInfo
- David Carpenter - IATI / AidInfo / AidView Database
- James Hughes - Kainos
- Alex Jackson - GDS Innovation team (Morning only)
- Paul Downey — GDS Technical Architect (Morning only)
- Mark Brough - Publish What You Fund (Morning only)
- **Online:** Siem Vaessen - Zimmerman & Zimmerman/Akvo - OIPA (Unable to join)
- **Online:** Dev Gateway - IATI Explorer - Dan Mihala (Unable to join)

Modality

Roundtable workshop, with small group discussions and direct work into an online document.

Resources

Post-its and flip-charts

Agenda

09.30-10.00	Arrival and coffee ● Groundrules: lightweight; agile; independent.
10.00-11.00	Existing APIs and Use Cases **(Phone in active)** ● What APIs exist?

	• Who is using it? See below
11.00-13.00	Reviewing current drafts; specific issue discussions. • Endpoints • Parameters & query formats • Response formats and options • JSON & CSV serialisations
13.00-14.00	Lunch
14.00-16.00	Break-out groups to focus on specific elements
16.00-17.00	Agreeing next steps • RFC to IATI TAG? •
17.00	**Pub?**

Existing/emerging APIs

X = Status from 0 = idea; 10 = full production tool.

API name	Data store	Output	Notes	N
IATI Explorer	eXist	XML	XPath access; Also Restful API declared.	6
OpenSpending (IATI dataset)	PostgreSQL	JSON, CSV	Flattens all activities down to transactions, and then breaks transactions down, creating one row per transaction per sector. Provides aggregations API and search API (using SOLR or ElasticSearch I think?) Aggregation: http://openspending.org/api/2/aggregate?dataset=iati&cut=time.year:2012|transaction_type.name:d&drilldown=from Transaction-level: http://openspending.org/api/2/search?dataset=iati	

			Background / briefing: http://openspending.org/resources/iati/index.html	
DFID Aid Information Platform	Graph Neo4J	JSON	Parsing into Neo4J; Running processes on import to roll up budgets; DFID specific at the moment; **Activity JSON** https://gist.github.com/5a6d4c54f4ee55c9ddad	5
OIPA (Github: https://github.com/openaid-IATI/OIPA)	MySQL	JSON/XML	Maps IATI xml to MySQL table for each namespace and subfields. Filtering enabled on some namespaces. Split by /activities/ and /organisation/ **JSON - all activities** http://oipa.openaidsearch.org/api/v3/activities/?format=json **XML - all activities** http://oipa.openaidsearch.org/api/v3/activities/?format=xml **JSON - single activity** http://oipa.openaidsearch.org/api/v3/activities/?format=json&iati_identifier=GB-1-202637 **JSON - single organisation** http://oipa.openaidsearch.org/api/v3/activities/?format=json&reporting_organisation__in=NL-1 **JSON - sectors** http://oipa.openaidsearch.org/api/v3/activities/?format=json&sectors__in=410,12220 **JSON - Joint call** http://oipa.openaidsearch.org/api/v3/activities/?format=json&reporting_organisation__in=NL-1&countries=BD&sectors_in=15160	7
AidView	eXist		http://datadev.aidinfolabs.org/data/ **XML - single activity** http://datadev.aidinfolabs.org/xquery/woapi.xq?ID=GB-1-202637&search=&start-date=&end-date=&start=1&pagesize=1000&result=full&format=xml&callback=&corpus=test&test=yes	7

			JSON - single activity http://datadev.aidinfolabs.org/xquery/woapi.xq?ID=GB-1-202637&search=&start-date=&end-date=&start=1&pagesize=1000&result=full&format=json&callback=&corpus=test&test=yes **Query API** http://datadev.aidinfolabs.org/xquery/woapi.xq?result=help **Spend by country for given organisation - JSON** http://datadev.aidinfolabs.org/xquery/woapi.xq?Funder=GB-CHC-1075920&ID=&search=&start-date=&end-date=&transaction=&groupby=Country&orderby=value&start=1&pagesize=1000&result=values&format=json&callback=&corpus=test&test=yes **- XML** http://datadev.aidinfolabs.org/xquery/woapi.xq?Funder=GB-CHC-1075920&ID=&search=&start-date=&end-date=&transaction=&groupby=Country&orderby=value&start=1&pagesize=1000&result=values&format=xml&callback=&corpus=test&test=yes	
LinkedData			SPARQL Access	5
CartoDB				
OKF - Registry Data Extra and Reporting Layer	PostgreSQL	?	?	
IATI Registry	CKAN	XML	Basic source for all IATI activities (grouped into country, region, non-regional files). Links to IATI activity and country files.	
IATI Preview		CSV	Challenges are around the different ways to flatten XML to CSV. Needs to be driven by how the data is used.	

User stories for APIs

From workshop held at GDS in July 2012
https://docs.google.com/spreadsheet/ccc?key=0ApHg07pak9cGdElPeGE4QmRXbHlNVGg0dGxLVzJ0MWc

Resource document

http://wiki.iatistandard.org/wg/api/start

Some typical aggregations:

- Count of activities per country
    - Query parameters:
        - groupby=recipient-country
        - function=count
        - element=iati-activity
    - Returns:
        - <aggregate>
          <recipient-country='AF'/>
          <count>123</count>
          </aggregate>
    - Simple method = countActivitesByCountry(country)
- Budget per country
    - Query parameters
        - groupby=recipient-country
        - function=sum
        - element=budget
    - **Business logic**: A revised budget for the same period as an original budget should replace it.
    - Returns
        - <aggregate>
          <recipient-country='AF'/>
          <values>100,000</values>
          </aggregate>
    - Simple method = budgetByCountry(country,year)
- Country budget by quarter/year from org file
- Sum of project budgets for country x by quarter/year
- Sector groups per country
- Sum of results indicators by country
- Total spend for contractor Y during period X
    - Query parameters

        - groupby=receiver-org, period (year/qtr/month)
        - function=sum
        - element=transaction (D or E)
        - ?receiver-org={org name}
        - ?transaction_transaction-type=D|E
    - **Business logic:** Provide response for each available transaction type, unless a filter is given.
    - **Business logic:** Aggregation services may provide by month, qtr, year *of transaction* etc. Need to define what we mean by 201202, 2012Q1 and 2012.
    - Returns
        - <aggregate>
          <receiver-org>PWC</receiver-org>
          <spend @type='D'>20000</spend>
          <spend @type='E'>20000</spend>
                <period>
                    <value>QTR1</value>
                     <spend @type='D'>10000</spend>
                     <spend @type='E'>10000</spend>
                </period>
                <period>
                    <value>QTR2</value>
                     <spend @type='D'>10000</spend>
                     <spend @type='E'>10000</spend>
                </period>
          </aggregates>

        - Alternative XML:
            - <row>
                <dimension name='period'>2012Q1</dimension>
                <dimension name='type'>D</dimension>
                <dimension name='reciever-org'>PWC</dimension>
                <value>10000</value>
              </row>
              <row>
                <dimension name='period'>2012Q1</dimension>
                <dimension name='type'>E</dimension>
                <dimension name='reciever-org'>PWC</dimension>
                <value>10000</value>
              </row>
    - CSV representation

receiver-org	period	type	spend

PWC	2012Q1	D	10000
PWC	2012Q1	E	10000
PWC	2012Q2	D	10000
PWC	2012Q2	E	10000

    - Simple method = spendByReceiver(receiver-org,period,transaction_type)

Typical pattern: sum of something BY geography/org FOR period

/aggregate/activity

/aggregate/transactions

/aggregate/results

-

Common serialisations

Aggregate query responses

asd