OPTIMADE notes

Outcomes

1. API Specification Doc: https://docs.google.com/document/d/1BbIfJ1fhPrsxQbZ8hdYZCJoMNF2BcbYGtzgiPAS6pt4/edit
    a. Plébiscite:
        i. custom query parameter extensions? format to use?
            1. x_
            2. something else
            3. nothing at all

Timeline of deployment

1) **9th November 2016**: deadline for implementation of major changes to specification document
2) **23rd November 2016**: all comments submitted to mailing list on specification document
3) **30th November 2016**: freeze the specification document, implementation of API and tests commence
4) **23rd December 2016**: implementation and tests should be complete, start writing a paper about the specification

People responsible for sections of the API specification document

1) **Queries**: Rickard Armiento
2) **Filter**: Saulius Grazulis
3) **Response**: Gareth Conduit
4) **Terminology**: Fawzi Mohamed
5) **Coherence**: Alexander Dorsk
6) **Compatibility**: Omnes
7) **Tests**: Omnes

Resources for comparing fields

1. Some hackzzz to get fields from various DBs: https://github.com/adorsk/lorentz-get-db-fields
2. Venn Diagram tool: http://bioinfogp.cnb.csic.es/tools/venny/
3. Other comparison ideas:
    a. diffs
    b. fuzzy comparison (text ‘edit distance’)

Slides

- Alex Dorsk, “Schema, Identifiers, Access: Lessons Learned from the Aspuru-Guzik Group”
- COD: http://www.crystallography.net/cod/archives/2016/slides/2016-Leiden-CECAM/slides.pdf

Resources workflow systems

a. Pegasus: https://pegasus.isi.edu/
b. Kniendpoint and output formatsme: https://www.knime.org/ (Greg Landrum from RDKit works on this)

Minutes of Monday discussion “Available REST API”

1) Place political pressure on journals to allow data to be shared, so that it can be used for machine learning. Should we keep a list of journals that allow data to be shared?
2) We would like hierarchical database access
3) In our database we want indexed by structure, but not compatible with experimental, however good for computational results
4) Should agree on list of properties that we want the database to store and the API to return: structures, query access, timestamp, authorship, computational code / hardware / compiler, workflow, material properties, trustworthiness / validation, access control, data format, notification / curation, registry / interconnection
5) We should make a list of use cases to crystallize what we want an API to do, probably in tomorrow’s from a user’s point of view discussion group
6) Create working groups: structure (it was agreed that this is complicated, dimensional authority was a popular approach); and three others

Minutes of Tuesday discussion “The user point of view”

1) Funding agency makes us store data in publications. As cheap to recompute do we need to store run outputs, or just the input parameters?
2) Group A-F report: two types of query that must be served by the API: property query and an introspective query. Many types of users (scientist, experimentalist, students, teacher, funding agency). Agree on names for properties. Agree on scope (include experimental data, can a user write to a database). Need a dynamic API. Need a unified definition of trustworthiness so that if property is in several databases the user can be given best quality result.
3) Group G-K report: We need to agree how to link databases together. Query delivers list of identifiers, second query on an identifier returns the property. Use cases: if look up property from one database get a link to another database. Structure matching to X-ray data. If looking at bulk iron also get reports about iron surface calculations, or other analogous materials.

4) Group L-? report: Need to agree on property names. Query is as simple as possible -- for example chemical composition or fingerprints such as Bravais lattice, then return should be simple -- one structure and link to database where comes from to get rest of the data. We need to think hard about what an experimentalist would want from a database.
5) Group ?-Z report: We should support queries that already exist, must be inter-database, exchange functional, all materials for which … has been calculated. Intelligence in the response for example return the most appropriate material in a database for a mechanical engineer, or return a putative material that would require a further computational calculation. Answer should arrive within 0.1s. We should learn from questions that have already been asked to pre-prepare answers.
6) Further discussion: API needs to be an enabler of a common GUI, so a single website could query all databases. Will the API just signpost to other databases, be designed so will only be used by electronic structure scientists. Can the API support a bulk download of summarized material data for use in an easy-to-use website, such as the SkyScanner website.

Minutes of Tuesday discussion “Common data”

1) *NOMAD case study*: files are well understood, backup and batch processing available. Want the file format to provide a clear definition, simple exchange, and implicit relationships (result to input parameter). Would like to abstract away the storage method, maybe use XML and a metadata dictionary or informal definition. If units are not specified assume SI units. Web APIS often use JSON, but it is difficult to return large amounts of data. Want open access of everything, able to track origin, API should be used because it is convenient, not because it is the only option. Use REST API for bulk sharing e.g. NOMAD.
2) *CIF case study*: structured human readable format, but defined by exact grammar so also machine readable. Defined dictionary.
3) *aflow case study*: Suggestion to create a wikipage on which to share information about each project’s database
4) Exchange correlation functional table, everyone is invited to edit. Link https://docs.google.com/spreadsheets/d/1b0bVj2JUVRoUOXP9_KqAcI-6u5BG9QmfLeF3XFrQ2Go/edit#gid=120341676 People also asked to add other spreadsheets. Could the table be created automatically from the documentation of each database. A small working group could form the aforementioned table.
5) It was agreed that we should have a discussion about how to discuss the format of the metadata.
6) A wikipage would be created during the proceeding implementation session listing what should be tabulated, based on the template pages already in other databases. Would start on simple quantities. These will be agreed later.

Minutes of Wednesday discussion “Validation of the data”

1) First, the definition of validation, verification, accuracy, and precision needed to be clarified. The discussion proceeded based around “trustworthiness”.

2) The “Delta test” (Science 2016) suggested the “Delta Factor” as a measure to capture difference between codes predicting the same property (considered as precision or verification).
3) The *NOMAD Encyclopedia* (currently developed) will show the variance of data with changing functional. Further activities of NOMAD concern the machine-learning of error bars
4) *Clean energy project*: calibrate against experiment (considered as validation)
5) *Conduit*: machine learning
6) We need to agree on the definitions of properties such as bandgap so that we can compare between databases.
7) Difficult to compare inter-databases, possible to verify data intra-database.
8) API should provide all the metadata so that a scientist could determine the accuracy.

Proposal: pancake API

1) API must be expandable and versioned, able to query by unambiguous descriptors
2) Should be read only, tell what additional properties are available
3) Should not inform about quality or query by computed properties without translation layer
4) Each database implements minimal API, and feed into a federation service
5) Federation service to continue in the future
6) Agreed that will have a REST API

Minutes of “Minimal API”, response working group

Format

json

Example for material request

```
{
   “query”: { “representation”: { serialized_query (or just the query string) },
                  “api_version”: ….
                  “time_stamp”: “timestamp”,
                  “items_returned”: integer,
                  “items_available”: integer,
                  “last_id”: “string” },
   “resource”: { “db_name” : “...”, ... }, # other info related to the DB I’m querying, like db_version if available, ...
# List of items returned by the queries
   “items”: [
   {
      “type”: “structure”,
      “properties”:
           {“formula”: “Es2 O3”, # will follow the convention defined by the query group
```

```
            “future_property”: {} },
        “local_id”: “compulsory id”,
        “uri”: “http://example.db/structs/0001”, # if available
        “immutable_id”: “http://example.db/structs/0001@123”, # optional field,
        “last_modified”: “timestamp in ISO 8061”,
        “custom_properties”: {dictionary of keyed-values defined by the specific db - other
option is to prefix them all with a string like “x_”, see below}
    },
    … # other results
   ],
    “response_message”: “OK”,
    “response_code”: 200, if returns or 204 if empty return,
    “meta” : { “DB specific custom field, not commonly standardized, global to the query” }
}
```

Example for introspective query

```
{
    “query”: { “representation”: { serialized_query (or just the query string) },
                    “api_version”: …
                    “time_stamp”: “timestamp”,
                    “items_returned”: integer,
                    “items_available”: integer,
                    “last_id”: “string” },
    “resource”: { “db_name” : “...”, ... }, # other info related to the DB I’m querying, like
db_version if available, ...
# List of items returned by the queries
    “items”: [
    {
        “type”: “introspective dictionary”,
        “properties”:
             { “property_name”: { “units”: “kDa”, “description”: “Dictionary of standard properties
that could be returned by the database, default units, textual description” } },
        “custom_properties”:
             { “property_name”: { “units”: “kDa”, “description”: “Dictionary of custom properties
that could be returned by the database, default units, textual description” } },
     },
    … # other results
   ],
    “response_message”: “OK”,
    “response_code”: 200, if returns or 204 if empty return
    “meta” : { “DB specific custom field, not commonly standardized, global to the query” }
}
```

Example resources

```
{
```

```
 “query”: { “representation”: { serialized_query (or just the query string) },
                    “api_version”: …
                    “time_stamp”: “timestamp”,
                    “items_returned”: integer,
                    “items_available”: integer,
                    “last_id”: “string” },
    “resource”: { “db_name” : “...”, ... }, # other info related to the DB I’m querying, like
db_version if available, ...
# List of items returned by the queries
    “items”: [
    {
        “type”: “resource” ,
        “database”: “name of database”,
        “version”: “version number”,
        “url”: “URL”,
        “last_modified”: “timestamp in ISO 8061”,
        “citation”: “citation”,
        “contact”: { “name”: “...”, “email”: “...” },
        “license”: “license info, public domain, GPL”,
        “additional_information”: {custom, optional database pagination information},
     },
    … # other results
   ],
    “response_message”: “OK”,
    “response_code”: 200, if returns or 204 if empty return because no materials
    “meta” : { “DB specific custom field, not commonly standardized, global to the query” }
}
```

Failure / exceptions

1) Bad request, mis-typed URL (400)
2) User does not have permission (401)
3) Forbidden: the server understood the request but refuses to authorize it (403)
4) Database was not found (404)
5) Request timeout because it is taking too long to process the query (408)
6) The database has been moved (410)
7) The response is too large (413)
8) The request URI contains more than 2048 characters (414)
9) Asked for a non-existent keyword (418)
10) Database returned (200) but the translator failed (422)

Resources

1) Workflow systems
    a. Pegasus: https://pegasus.isi.edu/
    b. Knime: https://www.knime.org/ (Greg Landrum from RDKit works on this)

URL Format

Minutes of “Minimal API”, url format group

Two major options: restful document hierarchy vs. url arguments.

- Standardize on restful document hierarchy:
  - http://exampe.com/minapi/v1/structures
  - http://exampe.com/minapi/v1/structures/<id>
  - http://exampe.com/minapi/v1/structures/<id>/canonical_chemical_formula
  - http://exampe.com/minapi/v1/query <- POST query endpoint
- Standardize on url arguments interface:
  - http://exampe.com/minapi?v=1&entry=structure&query={" canononical_chemical_formula ": "SiO2"}&fields=id,url&format=json
  - http://exampe.com/minapi_query <- POST query endpoint w. same params.

Philosophical difference: Is this a query interface or a standardized ‘element hierarchy’? E.g., the google search engine uses url arguments. If you look at a google document, it is indexed as /document/d/<document_id>/. Is the minimal api more a ‘search engine’ or a standard for a ‘collection of elements’?

Technical arguments / advantages / disadvantages:

- Most (?) databases already have rest-type URL hierarchies into their elements. If the minimal API enforces a standardized element hierarchy, most databases will have two competing ones. Is this something everyone wants to agree to? Is it the role of the minimal API to standardize this?
- You can use REST-ful helper libraries that provide great support for element hierarchies, e.g., django.
  - One question is: what advantage do these libraries actually provide? I (Alex) think it they may not save developers much time. They may actually complicate the implementation.
- A standardized document hierarchy may seem awkward if it does not map well to the specific underlying database structures.
- Why not just standardize a get/post query endpoint, and let databases that want to map that to an element hierarchy?
- How to pass along access token to element hierarchy?

Endpoint position in URL hierarchy: cannot standardize except for endpoint name. Exposing that you implement the minimal API means listing your endpoint.

Arguments in get or post: (this has not been vetted by the working group yet)

- Version, `v` (Default = latest version)
- What to search for: `entry` (mandatory)
- Query string: `query` (as standardized by query working group, defaults to ‘find all’)
- Output selection, `fields` (if not part of query, defaults to ‘all relevant fields’)

- Output format options: `format` (as standardized by output format working group, defaults to output format working group standard (json?))
- Access token: `key` (defaults to 'no access token')

Introspective get interface, e.g., ***(actual output format standardized by output group)***

- **Ask for available output formats**
  http://exampe.com/minapi?introspection=format

  {"format": [
  {"id":"json"},
  {"id":"x_example.com_hdf5"},
  …
  ]}
    - Implementations MUST provide the standard output format as decided by the output format working group. (json?)
    - Implementations MAY provide other output formats, but unless specified as part of the minimal api, locally invented identifiers must be prefixed with x_<domain>_
- **Ask for available object types (entries) to search for:**
  http :// example.com/minapi?v=1& introspection=entry&queryformat=json

  {"entry": [
  {"id":"structure"},
  {"id":"x_example.com_calculation"},
  …
  ]}
    - queryformat: what format is going to be set to in the query. Defaults to standard format as decided by output format working group (json?)
    - For queryformat=json, implementations MUST provide
      - structure.
    - Implementations MAY provide
      - dump: complete dump of database
    - Implementations MAY provide other element types, but unless specified as part of the minimal api, locally invented identifiers must be prefixed with x_<domain>_
- **Ask for query-identifiers (properties) that can be used in query:**
  http://example.com/minapi?v=1&introspection=property&entry=structure&queryformat=json

  {"property": [
  {"id":"id"},
  {"id":" canonical_chemical_formula "},
  {"id":"x_example.com_bulkmodulus"},
  …
  ]}

    - entry: what entry to query properties for (mandatory)
      queryformat: what format is going to be set to in the query. Defaults to standard format as decided by output format working group (json?)
    - For queryformat=json, implementations MUST provide:
        - id
        - canonical_chemical_formula
    - Implementations MAY provide others, but unless specified as part of the minimal api, locally invented identifiers must be prefixed with x_<domain>_
- **Ask for output fields (fields) that can be requested in output:** http://example.com/minapi?v=1&introspection=field&entry=structure&queryformat=json

  {"field": [
  {"id":"id"},
  {"id":"canonical_chemical_formula "},
  {"id":"url"},
  {"id":"x_example.com_bulkmodulus"},
  {"id":"x_example.com_dos"},
  …
  ]}
    - entry: what entry-type to query fields for (mandatory)
      queryformat: what format is going to be set to in the query. Defaults to standard format as decided by output format working group (json?)
    - For queryformats=json, implementations MUST provide:
        - id
        - canonical_chemical_formula
        - url
    - Implementations MAY provide others, but unless specified as part of the minimal api, locally invented identifiers must be prefixed with x_<domain>_

Notes:

- queryformat parameter in introspective query allows to qualify output on what format is used for output. E.g., no wave functions in json.
- queryformat parameter in introspective query allows to qualify entries on what format is used for output. E.g., entry=dump only for format x_example.com_targz.

Proposed spec

Summary

We propose a URL with single category endpoints along the lines of:

www.example.com/[whatever]/[endpoint]/[query]

Endpoints

(mandatory) info

This endpoint provides information about the API including versioin number, available endpoints, etc.

(optional) all

The generic endpoint that returns anything that matches a general query (from any of the other endpoints).

(optional) calculations

An endpoint to return calculations.

(optional) structures

Here structures refers to any collection of atoms and possibly an unit cell. This can be a simple structure, a ‘material’, a structure type or anything else.

Versions

By default the URL has no version number, e.g.:

example.com/minapi/[endpoint]

but users are free to keep old versions alive in which case they should use the standard form:

example/com/minapi/v1.0/[endpoint]

Minutes of the query working group

2016-10-26 initial record by S.G.

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1. URL query length: 2048 (Google search on browser supported limits); not a strict limit;
2. The minimal API MUST permit uniform way to pass responsible user’s email and identification token for databases that require it:
   email=user@domain.org
   acctoken=db1:ASDF1234,db2:QWERTYUIOP4567890
3. Two types of queries – a query that returns a list of IDs and a query that

returns data;

4. If the query is empty, select all resources;
5. If there more than 1000 resources selected, return the first 1000 and indicate that there are more;
6. There should be a mechanism to query for the next 1000 and so on, and to find out when the query has exhausted all resources (“pagination”);

Filters

1. For now, queries are flat, supporting one level of “AND” and “OR” conjunctions;
2. Querying chemical composition:
    1. elements=Si,Al,O – the conjunction should be “AND”, all records pertaining to materials containing Si, Al **and** O, and possibly other elements, MUST be returned; use 'nelements=3' to specify **exactly** 3 elements; (element=”Si,Al,O” means you want structures with at least the 3 elements, and it must contain Si, Al AND O).
    2. nelements=4 – return only entities that have exactly 4 elements. (element=”Si,Al,O”&nelements=4 means you want structures with exactly 4 elements: Si, Al AND O AND a fourth different element). To request a range of ‘nelement’ values, use the ‘nelements>=2 AND nelements<=7’ syntax;
    e.g. Materials Project: {"nelements": {"$gt": 2}}.
    3. chemical_formula=SiO2 – the formula must be **unreduced**. Element names MUST be with proper capitalization (Si, not SI for “silicon”). The formula “O2Si” is the same as “SiO2”, i.e. the order in which elements are specified should not be significant. No spaces or separators allowed for now.
    4. formula_prototype=A2BC
    The formula prototype is obtained by sorting elements by the occurrence number in the **reduced** chemical formula and replace them with subsequent alphabet letters A, B, C and so on.
    5. *Comments:*
        1. *(adorsk) it may be useful to look at existing chem. formula query formats, such as* *http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html*

2. *(adorsk) How much of a query interface do you really need? Thinking as a programmer who would have to implement the query handler, this could be somewhat tricky to program. Given that there are not large numbers of records in question (<100k), you could also push the burden of filtering to the requestor. (download everything, filter yourself on the downloaded records). Or, have an aggregator service that gathers the records via the minimal API do implement the querying.*

3. format=json – 'json' is a default and MUST be supported; other formats MAY be supported:
   format=cif
   format=xml – should we settle on the schema?
   format=html
   format=xhtml
   format=csv – can be used to return just list of ids.
   format=compression(gzip)

4. columns=tcod:_cell_length_a,mp:band_gap

5. modification_date=2016-10-26 – the date should be in ISO format; date-time queries permitted (RFC 3339):
   modification_date=2016-10-26+16:44:38**%2B**03:00
   modification_date>2016-10-26
   modification_date<=2016-10-26

6. id=db/1234567
   id=cod/2000000
   id=cod/2000000@1234567
   id=nomad/L1234567890,tcod/30000000,tcod/30000000@1234567

   The IDs MUST be URL-safe; in particular, they MAY NOT contain commas. Reasonably short IDs are encouraged (not longer than 255 characters).

   ID could be for a mutable resource, or additional immutable ID (mutable ID + revision nr.), with an MD5/SHA1/SHA256/... checksums.

   ID should be given by >= or <=, consistent with other parameters

7. after_id=db/567890,cod/1000000,nomad/L12344567

returns the results *after* the row identified by the id given in start_id. Looking it up will give the position in the total order of the results one has to starts with.

If there are more results a link with after_id already set. From http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#docs: “The right way to include pagination details today is using the Link header introduced by

RFC 5988. An API that uses the Link header can return a set of ready-made links so the API consumer doesn't have to construct links themselves."

Query results MUST be returned in an absolute order.

8. A database MAY implement 'db_revision=12345' key, to specify a specific revision, if implemented this should be always passed in the response
9. A database MUST implement a 'limit=100', and it MUST return no more than a 100 items in this case, it MIGHT return less. The database MIGHT have a top limit of the database which cannot be increased
10. refeference_id=cod/2000000,icsd/1234567,doi:10.1134/xyz123
11. comments:
    1. (adorsk) possible example filter syntaxes:
        1. https://lucene.apache.org/core/2_9_4/queryparsersyntax.html
        2. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html

This group should have subcommittee discussing "filters", another discussing queries, and a documentation group.

Minutes of Thursday discussion "Beyond minimal API"

1) Facility to request new calculations
2) Trustworthiness flag / confidence flag for different materials and properties
3) Including experimental and machine learning data: data type should be tagged, minimal API will be open for use by any type of materials data
4) Show what data is not present
5) Allow data upload: standard format for uploading data, standard set of information that should be provided
6) Complex search queries: nested searches, data sorting
7) GUI: Can be implemented on top of common API; multiple GUIs possible; both online front ends and local downloadable applications; return "best-fit" material for a certain set of required properties
8) Structured data: convex hulls, band structures, etc., plottable by GUIs
9) Common data generation protocols
10) Authentication: provide an email or token (should be in minimal API)
11) Central authentication service (beyond minimal API)
12) Shared common data curation software
13) Handle points duplicated across different databases

Minutes of implementation session "Towards a common API"

1) Need detailed documentation

Request/response examples, illustrating use cases

Comments

1. (adorsk) I'm putting this here, expecting this to change. But it may be useful to include a section like this in the draft doc we produce.

Give me all records

1. Request:
2. `curl http://db.org/api?action=query`

3. Responses:
    i. Case: has records:
    ii.
```
{
    "records": {
        "total": 3046,
        "records": [
            {"type": "db.org:structure", "id": "db.org:structure:Pqasf1"},
            {"type": "db.org:molecule", "id": "db.org:molecule:23Zasf"},
            ...
            {"type": "db.org:structure", "id": "db.org:structure:234Vas"}
        ]
    }
}
```

    iii. Case: does not have records:
    iv.
```
{
    "records": {
        "total": 0,
        "records": []
    }
}
```

    v. ---

Give me all structures

1. Request:
2. `curl http://db.org/api?action=query&entity_type=db.org:structure`

3. Response
    i. Case: Knows what a structure is.
    ii. Case: Has structure records:

iii.
```
{
  "records": {
    "count": 3023,
    "records": [
      {"type": "db.org:structure", "id": "db.org:structure:Pqasf1"},
      {"type": "db.org:molecule", "id": "db.org:molecule:23Zasf"},
      ...
      {"type": "db.org:structure", "id": "db.org:structure:234Vas"}
    ]
  }
}
```

a. Case: Does not have structure records:

b.
```
{
    "records": {
        "total": 0,
        "records": []
    }
}
```

c. Case: Does not know what a structure is:

d.
```
{
    "records": {
        "total": 0,
        "records": []
    }
}
```

Give me all molecules

1. Request:
2. `curl http://db.org/api?action=query&entity_type=db.org:molecule`

3. Response: as per structures.

Give me records updated after 2016-10-26 05:30

1. Request:
2. `curl http://db.org/api?action=query&q=updated:gt_201610260530`

3. Response:
    i. Case: has matching records:

ii.
```
{
  "records": {
    "count": 3023,
    "records": [
      {"type": "db.org:structure", "id": "db.org:structure:Pqasf1"},
      {"type": "db.org:molecule", "id": "db.org:molecule:23Zasf"},
      ...
      {"type": "db.org:structure", "id": "db.org:structure:234Vas"}
    ]
  }
}
```

iii. Case: no matching records:

iv.
```
{
  "records": {
    "count": 0,
    "records": []
  }
}
```

Give me structures with melting point < 200 and 'Au' in the chemical formula

1. Request
2. `curl http://db.org/api?action=query&q=[entity_type:db.org:structure] [melting_point < 200] [formula_contains:Au]`

3. Response
   i. Case: has matching records:
   ii.
```
{
  "records": {
    "count": 3023,
    "records": [
      {"type": "db.org:structure", "id": "db.org:structure:Pqasf1"},
      ...
      {"type": "db.org:structure", "id": "db.org:structure:234Vas"}
    ]
  }
}
```

   iii. Case: no matching records:

   iv.
```
{
  "records": {
```

```
    "count": 0,
    "records": []
  }
}
```

Give me the fields 'field_1' and 'field_2' for all structures

1. Request:
2. `curl http://db.org/api?action=query&q=[entity_type:db.org:structure]&fields=field_1,field_2`

3. Response:
    i. Case: has structures.
        a. Case: DB has all requested fields in its schema.
        b.
```
{
  "records": {
    "count": 3023,
    "records": [
      {"type": "db.org:structure", "id": "db.org:structure:Pqasf1",
       "fields": {"field_1": "value_1", "field_2": "value_2"}},
      ...
      {"type": "db.org:structure", "id": "db.org:structure:Pqasf1",
       "fields": {"field_1": "value_1", "field_2": null}},
    ]
  }
}
```

        a. Case: DB only has 'field_1':
        b.
```
{
  "records": {
    "count": 3023,
    "records": [
      {"type": "db.org:structure", "id":
"db.org:structure:Pqasf1",
       "fields": {"field_1": "value_1", "field_2": null}},
      ...
      {"type": "db.org:structure", "id":
"db.org:structure:Pqasf1",
       "fields": {"field_1": "value_1", "field_2": null}},
    ]
  }
```

```
}
```

ii. Does not have structures:
iii.
```
{
  "records": {
    "count": 0,
    "records": []
  }
}
```

Give me the version of the API

1. Request:
2. `curl http://db.org/api?action=introspect&fields=api_version`

3. Response:
4. `{"api_version": 1.2}`

Give me the types of entities you know about

1. Request:
2. `curl http://db.org/api?action=introspect&fields=entity_types`

3. Response:
4. `{"entity_types": ['structure', 'molecule', 'db.org:custom_entity_type']}`

Give me the structure properties you know about for structures and molecules

1. Request:
2. `curl http://db.org/api?action=introspect&filter=[entity_type:structure,molecule]&fields=properties`

3. Response:
4.
```
{
  "properties": {
    "structure": {
      "properties": [
        {"property_id": "db.org:chemical_formula", "description": ...},
        ...
```

```
        {"property_id": "db.org:custom_structure_property", "description": ...}
      ]
    }
    "molecule": {
      {"property_id": "db.org:chemical_formula", "description": ...},
      ...
      {"property_id": "db.org:custom_molecule_property", "description": ...}
    }
  }
}
```

Minutes of the query working group

2016-10-26 initial record by S.G.

Minutes of Thursday discussion “Beyond minimal API”

1) Facility to request new calculations
2) Trustworthiness flag / confidence flag for different materials and properties
3) Including experimental and machine learning data: data type should be tagged, minimal API will be open for use by any type of materials data
4) Show what data is not present
5) Allow data upload: standard format for uploading data, standard set of information that should be provided
6) Complex search queries: nested searches, data sorting
7) GUI: Can be implemented on top of common API; multiple GUIs possible; both online front ends and local downloadable applications; return “best-fit” material for a certain set of required properties
8) Structured data: convex hulls, band structures, etc., plottable by GUIs
9) Common data generation protocols
10) Authentication: provide an email or token (should be in minimal API)
11) Central authentication service (beyond minimal API)
12) Shared common data curation software
13) Handle points duplicated across different databases

Minutes of implementation session “Towards a common API”

1) Need detailed documentation

---

1. after_id=db/567890,cod/1000000,nomad/L12344567

returns the results *after* the row identified by the id given in start_id. Looking it up

will give the position in the total order of the results one has to starts with.

If there are more results a link with after_id already set. From http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#docs: “The right way to include pagination details today is using the Link header introduced by RFC 5988. An API that uses the Link header can return a set of ready-made links so the API consumer doesn't have to construct links themselves.”

Query results MUST be returned in an absolute order.

2. A database MAY implement ‘db_revision=12345’ key, to specify a specific revision, if implemented this should be always passed in the response
3. refeference_id=cod/2000000,icsd/1234567,doi:10.1134/xyz123
4. comments:
    1. (adorsk) possible example filter syntaxes:
        1. https://lucene.apache.org/core/2_9_4/queryparsersyntax.html
        2. https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html

This group should have subcommittee discussing “filters”, another discussing queries, and a documentation group.