followthemoney

OCCRP

Sep 15, 2021

CONTENTS:

FollowTheMoney (FtM) is a data model for anti-corruption investigations. It contains definitions of the entities relevant in such research (like *people* or *companies*) and tools that let you generate, validate, and export such data easily. Entities can reference each other, thus creating a graph of relationships.

FtM can be used in three contexts: as a *command-line utility*, a *Python library*, and as a TypeScript/JavaScript library. The ontology defined by FtM also includes a model for various types of *documents* that might be used as evidence in investigations[1].

All data stored by the Aleph search engine is expressed as FtM entities. Aleph itself adds functions for searching, viewing, and manipulating such entities. It also introduces higher-level notions of datasets and access control.

[1] The Aleph toolchain includes a separate project, `ingestors`, which can extract and analyse the content of many document types and emit them as FtM entities. For example, if you were to process an archive of emails, it would generate a complex graph of E-Mail entities that connect the people sending and receiving them.

CHAPTER

ONE

INTRODUCTION

followthemoney (FtM) defines a simple data model for storing complex object graphs. You will need to understand three concepts: *entities*, *entity references*, and *entity streams*.

1.1 Entities

Entities are often expressed as snippets of JSON, with three standard fields: a unique `id`, a specification of the type of the entity called `schema`, and a set of `properties`. `properties` are multi-valued and values are always strings.

```
{
    "id": "1b38214f88d139897bbd13eabde464043d84bbf9",
    "schema": "Person",
    "properties": {
        "name": ["John Doe"],
        "nationality": ["us", "au"],
        "birthDate": ["1982"]
    }
}
```

Property names are defined by the *Schemata definitions*. For example, a *Person* has a *nationality*, while a *Company* allows for setting a *jurisdiction*. Both properties, however, have the same *property type*, *Country*.

1.2 References

Entities can reference other entities. This is achieved via a special property type, *Entity*. Properties of this type simply store the ID of another entity. For example, a *Passport* entity can be linked to a *Person* entity via its *holder* property:

```
{
    "id": "passport-entity-id",
    "schema": "Passport",
    "properties": {
        "holder": ["person-entity-id"],
        "number": ["CJ 7261817"]
    }
}
```

Note: Applications using *FtM* data usually need to resolve references bi-directionally. In the context of the example above, they will need to access the person based on it's ID in order to follow the *holder* link, but also query an inverted

index to retrieve all the passports which reference a given person.

In Aleph this is achieved using ElasticSearch and exposed via the `/api/2/entities/<id>/expand` API endpoint.

1.2.1 Interstitial entities

A link between two entities will have its own attributes. For example, an investigator looking at a person that owns a company might want to know when that interest was acquired, and also what percentage of shares the person holds.

This is addressed by making interstitial entities. In the example above, an *Ownership* entity would be created, with references to the person as its *owner* property and to the company as its *asset* property. That entity can then define further properties, including *startDate* and *percentage*:

```
{
    "id": "ownership-entity-id",
    "schema": "Ownership",
    "properties": {
        "owner": ["person-entity-id"],
        "asset": ["company-entity-id"],
        "startDate": ["2020-01-01"],
        "percentage": ["51%"],
    }
}
```

Warning: It is tempting to simplify this model by assuming that entities derived from *Thing* are node entities, and those derived from *Interval* are edges. This assumption is false and will lead to nasty bugs in your code.

1.3 Streams

Many tools in the *FtM* ecosystem use streams of entities to transfer or store information. Entity streams are simply sequences of entity objects that have been serialised to JSON as single lines without any indentation, each entity separated by a newline.

Entity streams are read and produced by virtually every part of the *Command-line functions*, the Aleph API, and they're also supported by the *ingestors*. When stored to disk as a file, the extensions *.ftm* or *.ijson* should be used.

CHAPTER

TWO

ENTITY AND SCHEMA API

The core interfaces of *followthemoney* are simple: each running instance of the library has a *Model* singleton, which holds a set of *Schema* definitions (e.g. *Person*). Each schema defines a set of *properties* (e.g. *name*, *birthDate*) which give meaning to how values can be associated with entities of a given schema.

The *model* is also used to instantiate *entity proxies* - objects that allow the creation and use of entity data, based on the rules defined by an associated *schema*.

2.1 Example

For an illustration of how these objects interact, imagine the following script:

```
# Load the standard instance of the model
from followthemoney import model

## Schema metadata
# Access a schema metadata object
schema = model.get('Person')

# Access a property metadata object
prop = schema.get('birthDate')

# You can also import the type registry that lets you access type info easily:
from followthemoney.types import registry
assert prop.type == registry.date

## Working with entities and entity proxies
# Next, let's instantiate a proxy object for a new Person entity:
entity = model.make_entity(schema)

# First, you'll want to assign an ID to the entity. You can do this directly:
entity.id = 'john-smith'

# Or you can use a hashing function to make a safe ID:
entity.make_id('John Smith', '1979')

# Now, let's assign this entity a birthDate property (see above):
entity.add(prop, '1979-08-23')

# You can also assign properties by name:
```

(continues on next page)

(continued from previous page)

```
entity.add('firstName', 'John')
entity.add('lastName', 'Smith')
entity.add('name', 'John Smith')

# Adding a property value will perform some validation:
entity.add('nationality', 'Atlantis')
assert not entity.has('nationality')
entity.add('nationality', 'Germani', fuzzy=True)
assert 'de' == entity.first('nationality')

# Lets make a second entity, this time for a passport:
passport_entity = model.make_entity('Passport')
passport_entity.make_id(entity.id, 'C716818')
passport_entity.add('number', 'C716818')

# Entities can link to other entities like this:
passport_entity.add('holder', entity)
# Which is the same as:
passport_entity.add('holder', entity.id)

# Finally, you can turn the contents of the entity proxy into a plain dictionary
# that is suitable for JSON serialization or storage in a database:
data = entity.to_dict()
assert data.get('id') == entity.id

# If you want to turn this back into an entity proxy:
entity2 = model.get_proxy(data)
assert entity2 == entity
```

The library offers a much more complex set of operations - but entity proxies, schemata, properties, and the model are the key elements to understand.

2.2 Entity proxy

The entity proxy is a wrapper object for FtM data. It can be used as a factory in order to build an entity, or as a simple abstraction to query the properties of an existing entity.

class followthemoney.proxy.**EntityProxy**(*model*, *data*, *key_prefix=None*, *cleaned=True*)

A wrapper object for an entity, with utility functions for the introspection and manipulation of its properties.

This is the main working object in the library, used to generate, validate and emit data.

add(*prop*, *values*, *cleaned=False*, *quiet=False*, *fuzzy=False*)

Add the given value(s) to the property if they are valid for the type of the property.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **values** – either a single value, or a list of values to be added.
- **cleaned** – should the data be normalised before adding it.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

- **fuzzy** – when normalising the data, should fuzzy matching be allowed.

property caption
The user-facing label to be used for this entity. This checks a list of properties defined by the schema (caption) and returns the first available value. If no caption is available, return the schema label.

clone()
Make a deep copy of the current entity proxy.

context
If the input dictionary for the entity proxy contains fields other than `id`, `schema` or `properties`, they will be kept in here and re-added upon serialization.

property countries
Get the set of all country-type values set of the entity.

property country_hints
Some property types, such as phone numbers and IBAN codes imply a country that may be associated with the entity. This list can be used for a more generous matching approach than the actual country values.

edgepairs()
Return all the possible pairs of values for the edge source and target if the schema allows for an edge representation of the entity.

first(*prop*, *quiet=False*)
Get only the first value set for the property, in no particular order.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

Returns A value, or `None`.

classmethod from_dict(*model*, *data*, *cleaned=True*)
Instantiate a proxy based on the given model and serialised dictionary.

Use *followthemoney.model.Model.get_proxy()* instead.

get(*prop*, *quiet=False*)
Get all values of a property.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

Returns A list of values.

get_type_inverted(*matchable=False*)
Return all the values of the entity arranged into a mapping with the group name of their property type. These groups include `countries`, `addresses`, `emails`, etc.

get_type_values(*type_*, *matchable=False*)
All values of a particular type associated with a the entity. For example, this lets you return all countries linked to an entity, rather than manually checking each property to see if it contains countries.

Parameters

- **type** – The type object to be searched.

- **matchable** – Whether to return only property values marked as matchable.

has(*prop*, *quiet=False*)
Check to see if the given property has at least one value set.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

Returns a boolean.

id
A unique identifier for this entity, usually a hashed natural key, a UUID, or a very simple slug. Can be signed using a *Namespace*.

iterprops()
Iterate across all the properties for which a value is set in the proxy (but do not return their values).

itervalues()
Iterate across all values in the proxy one by one, each given as a tuple of the property and the value.

key_prefix
When using *make_id()* to generate a natural key for this entity, the prefix will be added to the ID as a salt to make it easier to keep IDs unique across datasets. This is somewhat redundant following the introduction of *Namespace*.

make_id(**parts*)
Generate a (hopefully unique) ID for the given entity, composed of the given components, and the *key_prefix* defined in the proxy.

merge(*other*)
Merge another entity proxy into this one. This will try and find the common schema between both entities and then add all property values from the other entity into this one.

property names
Get the set of all name-type values set of the entity.

pop(*prop*, *quiet=True*)
Remove all the values from the given property and return them.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

Returns a list of values, possibly empty.

property properties
Return a mapping of the properties and set values of the entity.

remove(*prop*, *value*, *quiet=True*)
Remove a single value from the given property. If it is not there, no action takes place.

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **value** – will not be cleaned before checking.

- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

schema
The schema definition for this entity, which implies the properties That can be set on it.

set(*prop*, *values*, *cleaned=False*, *quiet=False*)
Replace the values of the property with the given value(s).

Parameters

- **prop** – can be given as a name or an instance of *Property*.
- **values** – either a single value, or a list of values to be added.
- **cleaned** – should the data be normalised before adding it.
- **quiet** – a reference to an non-existent property will return an empty list instead of raising an error.

to_dict()
Serialise the proxy into a dictionary with the defined properties, ID, schema and any contextual values that were handed in initially. The resulting dictionary can be used to make a new proxy, and it is commonly written to disk or a database.

to_full_dict(*matchable=False*)
Return a serialised version of the entity with inverted type groups mixed in. See *get_type_inverted()*.

triples(*qualified=True*)
Serialise the entity into a set of RDF triple statements. The statements include the property values, an `RDF#type` definition that refers to the entity schema, and a `SKOS#prefLabel` with the entity caption.

2.3 Schema management

class followthemoney.schema.**Schema**(*model*, *name*, *data*)
A type definition for a class of entities that have certain properties.

Schemata are arranged in a multi-rooted hierarchy: each schema can have multiple parent schemata from which it inherits all of their properties. A schema can also have descendant child schemata, which, in turn, add further properties. Schemata are usually accessed via the model, which holds all available definitions.

abstract
Do not store or emit entities of this type, it is used only for inheritance.

caption
Mark a set of properties to be used for the entity's caption. They will be checked in order and the first existant value will be used.

descendants
Inverse of *schemata*, all derived child types of this schema and their children.

property description
A longer description of the semantics of the schema.

edge
Flag to indicate if this schema should be represented by an edge (rather than a node) when the data is converted into a property graph.

edge_directed
Flag to indicate if the edge should be presented as directed to the user, e.g. by showing an error at the target end of the edge.

property edge_label
Description label for edges derived from entities of this schema.

extends
Direct parent schemata of this schema.

featured
Mark a set of properties as important, i.e. they should be shown first, or in an abridged view of the entity. In Aleph, these properties are included in tabular entity listings.

generate()
While loading the schema, this function will validate and load the hierarchy, properties, and flags of the definition.

generated
Entities with this type are generated by the system - for example, via *ingest-file*. The user should not be offered an option to create them in the interface.

get(*name*)
Retrieve a property defined for this schema by its name.

hidden
Hide this schema in listings.

is_a(*other*)
Check if the schema or one of its parents is the same as the given candidate `other`.

property label
User-facing name of the schema.

matchable
Try to perform fuzzy matching. Fuzzy similarity search does not make sense for entities which have a lot of similar names, such as land plots, assets etc.

property matchable_schemata
Return the set of schemata to which it makes sense to compare with this schema. For example, it makes sense to compare a legal entity with a company, but it does not make sense to compare a car and a person.

name
Machine-readable name of the schema, used for identification.

names
All names of *schemata*.

property plural
Name of the schema to be used in plural constructions.

properties
The full list of properties defined for the entity, including those inherited from parent schemata.

required
Mark a set of properties as required. This is applied only when an entity is created by the user - bulk created entities will slip through even if it is technically invalid.

schemata
All parents of this schema (including indirect parents and the schema itself).

property sorted_properties
All properties of the schema in the order in which they should be shown to the user (alphabetically, with captions and featured properties first).

property source_prop
The entity property to be used as an edge source.

property target_prop
The entity property to be used as an edge target.

to_dict()
Return schema metadata, including all properties, in a serializable form.

uri
RDF identifier for this schema when it is transformed to a triple term.

validate(*data*)
Validate a dictionary against the given schema. This will also drop keys which are not valid as properties.

class followthemoney.property.**Property**(*schema*, *name*, *data*)
A definition of a value-holding field on a schema. Properties define the field type and other possible constraints. They also serve as entity to entity references.

RESERVED = ['id', 'caption', 'schema', 'schemata']
Invalid property names.

property description
A longer description of the semantics of this property.

generate()
Setup method used when loading the model in order to build out the reverse links of the property.

hidden
This property should not be shown or mentioned in the user interface.

property label
User-facing title for this property.

matchable
Whether this property should be used for matching and cross-referencing.

name
Machine-readable name for this property.

qname
Qualified property name, which also includes the schema name.

range
If the property is of type `entity`, the set of valid schema to be added in this property can be constrained. For example, an asset can be owned, but a person cannot be owned.

reverse
When a property points to another schema, a stub reverse property is added as a place to store metadata to help display the link in inverted views.

schema
The schema which the property is defined for. This is always the most abstract schema that has this property, not the possible child schemata that inherit it.

specificity(*value*)
Return a measure of how precise the given value is.

stub
When a property points to another schema, a reverse property is added for various administrative reasons. These properties are, however, not real and cannot be written to. That's why they are marked as stubs and adding values to them will raise an exception.

to_dict()
Return property metadata in a serializable form.

type
The data type for this property.

uri
RDF term for this property (i.e. the predicate URI).

validate(*data*)
Validate that the data should be stored.

Since the types system doesn't really have validation, this currently tries to normalize the value to see if it passes strict parsing.

class followthemoney.model.**Model**(*path*)
A collection of all the schemata available in followthemoney. The model provides some helper functions to find schemata, properties or to instantiate entity proxies based on the schema metadata.

common_schema(*left*, *right*)
Select the most narrow of two schemata.

When indexing data from a dataset, an entity may be declared as a LegalEntity in one query, and as a Person in another. This function will select the most specific of two schemata offered. In the example, that would be Person.

generate()
Loading the model is a weird process because the schemata reference each other in complex ways, so the generation process cannot be fully run as schemata are being instantiated. Hence this process needs to be called once all schemata are loaded to finalise dereferencing the schemata.

get(*name*)
Get a schema object based on a schema name. If the input is already a schema object, it will just be returned.

get_proxy(*data*, *cleaned=True*)
Create an entity proxy to reflect the entity data in the given dictionary. If `cleaned` is disabled, all property values are fully re-validated and normalised. Use this if handling input data from an untrusted source.

get_qname(*qname*)
Get a property object based on a qualified name (i.e. schema:property).

get_type_schemata(*type_*)
Return all the schemata which have a property of the given type.

make_entity(*schema*, *key_prefix=None*)
Instantiate an empty entity proxy of the given schema type.

make_mapping(*mapping*, *key_prefix=None*)
Parse a mapping that applies (tabular) source data to the model.

map_entities(*mapping*, *key_prefix=None*)
Given a mapping, yield a series of entities from the data source.

properties
All properties defined in the model.

schemata
A mapping with all schemata, organised by their name.

to_dict()
Return metadata for all schemata and properties, in a serializable form.

2.4 Helper utilities

followthemoney.helpers.**entity_filename**(*proxy*, *base_name=None*, *extension=None*)
Derive a safe filename for the given entity.

followthemoney.helpers.**inline_names**(*entity*, *related*)
Attempt to solve a weird UI problem. Imagine we are showing a list of payments between a sender and a beneficiary to a user. They may now conduct a search for a term present in the sender or recipient name, but there will be no result, because the name is only indexed with the parties, but not in the payment. This is part of a partial work-around to that.

This is really bad in theory, but really useful in practice. Shoot me.

followthemoney.helpers.**name_entity**(*entity*)
If an entity has multiple names, pick the most central one and set all the others as aliases. This is awkward given that names are not special and may not always be the caption.

followthemoney.helpers.**remove_checksums**(*proxy*)
When accepting entities via a web API, it would consistute a security risk to allow a user to submit checksum-type properties. These can be traded in for access to said files if they exist in the underlying content-addressed storage. It seems safest to just remove all checksums from entities when they are untrusted user input.

followthemoney.helpers.**remove_prefix_date_values**(*values*)
See remove_prefix_dates.

followthemoney.helpers.**remove_prefix_dates**(*entity*)
If an entity has multiple values for a date field, you may want to remove all those that are prefixes of others. For example, if a Person has both a birthDate of 1990 and of 1990-05-01, we'd want to drop the mention of 1990.

followthemoney.helpers.**simplify_provenance**(*proxy*)
If there are multiple dates given for some of the provenance fields, we can logically conclude which one is the most meaningful.

followthemoney.util.**get_entity_id**(*obj: Any*) → Optional[str]
Given an entity-ish object, try to get the ID.

followthemoney.util.**key_bytes**(*key: Any*) → bytes
Convert the given data to a value appropriate for hashing.

followthemoney.util.**merge_context**(*left: Dict[followthemoney.util.K, followthemoney.util.V]*, *right: Dict[followthemoney.util.K, followthemoney.util.V]*) → Dict[followthemoney.util.K, List[followthemoney.util.V]]
When merging two entities, make lists of all the duplicate context keys.

CHAPTER

THREE

ENTITY FRAGMENTATION

Generating graph data is a difficult process. The size of the datasets we want to process using *followthemoney* (FtM) makes it impossible to incrementally build nodes and egdes in memory like you would in NetworkX. Instead, we use a stream-based solution for constructing graph entities. That is why the toolkit supports *entity fragments* and *aggregation*.

3.1 An example

To illustrate this problem, imagine a table with millions of rows that describes a set of people and the companies they control. Every company can have multiple directors, while each director might control multiple companies:

CompanyID	CompanyName	DirectorName	DirectorIdNo	DirectorDoB
A123	Brilliant Amazing Ltd.	John Smith	PP827817	1979-02-16
…	…	…	…	…
A71882	Goldfish Ltd.	John Smith	PP827817	NULL
…	…	…	…	…
A123	Brilliant Amazing Ltd.	Jane Doe	PP1988299	1983-06-24
…	…	…	…	…

3.2 Database humpty-dumpty

When *turning this data into FtM*, we'd create three entities for each row: a *Company*, a *Person* and a *Directorship* that connects the two.

If we do this row by row, we'd eventually generate three *Company* entities to represent two actual companies, and three *Person* entities for two distinct people. Of course, we could write these to an ElasticSearch index sequentially - the later entities overwriting the earlier ones with the same ID.

That works only as long as each version of each entity contains the same data. In our example, the first mention of *John Smith* includes his birth date, while the second does not. If we don't wish to lose that detail, we need to merge these *fragments*. While it's possible to perform such merges at index time, this has proven to be impractically slow because it requires fetching each entity before it is updated.

A better solution is to sort all generated fragments before indexing them. With this approach, all the entities generated from the source table would be written to disk or to a database, and then sorted using their ID. In the resulting entity set, all instances of each company and person are grouped and can be merged as they are read.

3.3 In practice

In the FtM toolchain, there are two tools for doing entity aggregation: from the command-line *ftm aggregate* will merge fragments in memory. Alternately the add-on library *followthemoney-store* will perform the same operation in a SQLite or PostgreSQL database.

```
# Generate entities from a CSV file and a mapping:
cat company-registry.csv | ftm map-csv mapping_file.yml >fragments.ijson
# Write the fragments to a table `company_registry`:
cat fragments.ijson | ftm store write -d company_registry
# List the tables in the store:
ftm store list
# Output merged entities:
ftm store iterate -d company_registry
```

The same functionality can also be used as a Python library:

```
import os
from ftmstore import get_dataset
# Assume a function that will emit fragments:
from myapp.data import generate_fragments

# If no `database_uri` is given, ftmstore will read connection from
# $FTM_STORE_URI, or create a file called `followthemoney.sqlite` in
# the current directory.
database_uri = os.environ.get('DATABASE_URI')
dataset = get_dataset('myapp_dataset', database_uri=database_uri)
bulk = dataset.bulk()
for idx, proxy in enumerate(generate_fragments()):
    bulk.put(proxy, fragment=idx)
bulk.flush()

# This will print the number of combined entities (ie. DISTINCT id):
print(len(dataset))

# This will return combined entities:
for entity in dataset.iterate():
    print(entity.caption)

# You could also iterate the underlying fragments:
for proxy in dataset.partials():
    print(proxy)

# Note: `dataset.partials()` returns `EntityProxy` objects. The method
# `dataset.fragments()` would return raw Python dictionaries instead.

# All three methods also support the `entity_id` filter, which can also be
# shortened to `get`:
entity = dataset.get(entity_id)
```

3.4 Fragment origins

followthemoney-store is used across the tools built on FtM to capture and aggregate entity fragments. In Aleph, fragments for one entity might be written by different processes: the API, document ingestors, document NER analyzers or a translation backend. It is convenient to be able to flush all entity fragments from a particular origin, while leaving the other fragments intact. For example, this can be used to delete all data uploaded via the bulk API, while leaving document-based data in the same dataset intact.

To support this, *ftm-store* has the notion of an *origin* for each fragment. If specified, this can be used to later delete or overwrite subsets of fragments.

```
cat us_ofac.ijson | ftm store write -d sanctions -o us_ofac
cat eu_eeas.ijson | ftm store write -d sanctions -o eu_eeas

# Will now have entities from both source files:
ftm store iterate -d sanctions | wc -l

# Delete all fragments from the second file:
ftm store delete -d sanctions -o eu_eeas

# Only one source file is left:
ftm store iterate -d sanctions | wc -l
```

CHAPTER

FOUR

NAMESPACING

We like our abstractions like our offshore banks: leaky.

Entity ID namespaces are a security mechanism related to the Aleph search index.

Aleph allows the user (via mappings or the API) to create arbitary entity IDs. Entity IDs that are controlled by the user and not the system are unusual. However, this makes it possible to generate bulk data outside Aleph, and then load entities into the system as a continuous *Streams*.

The problem is that having user controlled entity IDs increases the chance of conflict in the search index.

Namespacing works around this by making each entity ID consist of two parts: one controlled by the client, the other controlled by the system. The second part of the ID is called its *signature*:

```
entity_id.a40a29300ac6bb79dd2f911e77bbda7a3b502126
```

The signature is generated as `hmac(entity_id, dataset_id)`. This guarantees that the combined ID is specific to a dataset, without needing an (expensive) index look up of each ID first. It can also be generated on the client or the server without compromising isolation.

class followthemoney.namespace.**Namespace**(*name=None*)

Namespaces are used to partition entity IDs into different units, which traditionally represent a dataset, collection or source.

See module docstring for details.

SEP = '.'

apply(*proxy*, *shallow=False*)

Rewrite an entity proxy so all IDs mentioned are limited to the namespace.

classmethod make(*name*)

classmethod parse(*entity_id*)

Split up an entity ID into the plain ID and the namespace signature. If either part is missing, return None instead.

sign(*entity_id*)

Apply a namespace signature to an entity ID, removing any previous namespace marker.

signature(*entity_id*)

Generate a namespace-specific signature.

classmethod strip(*entity_id*)

verify(*entity_id*)

Check if the signature matches the current namespace.

CHAPTER

FIVE

MAPPINGS

Mappings are a mechanism for generating entities from structured data sources, including tabular data and SQL databases.

Please refer to the Aleph mappings documentation for details.

CHAPTER

SIX

EXTENDING THE SCHEMA

followthemoney (FtM) lives a double life, as a data modelling and validation tool, and as a specific data model. While the default model is tailored to investigative journalism, the data modelling tool is reusable beyond that domain. Users from adjacent fields, such as human rights advocacy, cyber forensics or social media analysis might wish to extend the data model.

6.1 Modelling principles

The goal of *FtM* is to create a data model for international business, banking, corporate law, government procurement, and social relations. It's important to be conscious of how preposterous that premise is. In consequence, our goal cannot be to fully model these domains, but to *model a conversation between two slightly drunk investigative reporters* about these things.

Accordingly, the following heuristics should be applied to schema extensions:

- Does the schema change solve a practical precision issue in a dataset or in a client application, or will it merely serve to better reflect a legal theory?
- Can the framing of entities and their relations be expressed in such a way that they are applicable to more than one jurisdiction? (Jurisdiction-specific properties are OK)
- Avoid modelling out schema without different properties: we don't need *Doctor*, *Lawyer*, *TruckDriver*, *Farmer* as schemata, when this can just be a *profession* property.

6.2 Schema syntax

For a reference of how schema definitions are designed, use the following resources:

- Check out the existing YAML files in `followthemoney/schema` in the source repository.
- The documentation for *followthemoney.schema.Schema* explains the options available on for defining schema metadata.
- The documentation for *followthemoney.property.Property* gives more detail regarding property descriptors.

6.3 Instance-specific overrides

Operators can modify the active FtM model by setting the environment variable `$FTM_MODEL_PATH` to a directory that contains an alternate set of YAML schema description files. The simplest way to do this would be to download the existing YAML schema set from FtM source (*followthemoney/schema*), and make edits in a new directory.

This raises an difficult question: how mutable is the FtM model in reality? FtM uses specific schemata in its tests, but that wouldn't be a fatal hindrance. However, the *aleph* and *ingestors* applications have built-in assumptions about a larger subsection of the schema. For example, *ingestors* require all schemata descendant from *File* and *Mention*. Aleph assumes that the *Legal entity*, *Person* and *File* types exist.

In practice, we would recommend these policies when modifying the FtM model in an Aleph deployment:

- Add schemata and properties, don't remove or rename the existing ones.
- Don't add too many schemata. Each schema will add a new index in Aleph's ElasticSearch, and each index needs to be queried individually. Add 10 or 20 schemata, not 200.
- Contribute upstream: once you have successfully modelled some data into an extension of FtM, please feel free to submit a pull request to FtM to have them adopted into the main repo.

CHAPTER

SEVEN

COMMAND-LINE FUNCTIONS

Many of the functions of *followthemoney* (FtM) can be used interactively or in scripts via the command line. Please first refer to the Aleph documentation for an intro to the `ftm` utility.

Key to understanding the `ftm` tool is the notion of *Streams*: entities can be transferred between programs and processing steps as a series of JSON objects, one per line. This notion is supported by the related alephclient command, which can serve as a source, and a sink for entity streams, backed by the Aleph API.

7.1 Examples

The command line sequence below uses shell pipes to a) *map data* into entities from a database, b) apply a *namespace* to the entity IDs, c) aggregate *entity fragments* created by the mapping, and d) export the resulting entity stream into a sequence of CYPHER statements that can be executed on a Neo4J database to generate a property graph:

```
ftm map companies_from_db.yml | \
    ftm sign -s my_namespace | \
    ftm aggregate | \
    ftm export-cypher -o graph.cypher
```

Here's another example that fetches pre-generated entities from a URL and loads them into a local Aleph instance:

```
export URL=https://public.data.occrp.org/datasets/icij/panama_papers.ijson
curl -s $URL | \
    ftm validate | \
    alephclient write-entities -f icij_panama_papers
```

7.2 Command reference

7.2.1 ftm

Utility for FollowTheMoney graph data

```
ftm [OPTIONS] COMMAND [ARGS]...
```

aggregate

Aggregate multiple fragments of entities

```
ftm aggregate [OPTIONS]
```

Options

-i, **--infile** <infile>

-o, **--outfile** <outfile>

dump-model

Export the current schema model

```
ftm dump-model [OPTIONS]
```

Options

-o, **--outfile** <outfile>

export-csv

Export to CSV

```
ftm export-csv [OPTIONS]
```

Options

-i, **--infile** <infile>

-o, **--outdir** <outdir>
output directory

export-cypher

Export to Cypher script

```
ftm export-cypher [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-e, --edge-types <edge_types>
Property types to be reified into graph edges.

Options iban | identifier | country | url | domain | date | ip | phone | checksum | entity | address | name | email

export-excel

Export to Excel

```
ftm export-excel [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>
Required

export-gexf

Export to GEXF (Gephi) format

```
ftm export-gexf [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-e, --edge-types <edge_types>
Property types to be reified into graph edges.

Options iban | identifier | country | url | domain | date | ip | phone | checksum | entity | address | name | email

export-neo4j-bulk

Export to Neo4J bulk import

```
ftm export-neo4j-bulk [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outdir <outdir>
output directory

-e, --edge-types <edge_types>
Property types to be reified into graph edges.

Options iban | identifier | country | url | domain | date | ip | phone | checksum | entity | address | name | email

export-rdf

Export to RDF NTriples

```
ftm export-rdf [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

--qualified, --unqualified
Generate full predicates

import-vis

Load a .VIS file and get entities

```
ftm import-vis [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

link

Apply matches from a deduplication match file

```
ftm link [OPTIONS]
```

Options

-i, **--infile** <infile>

-o, **--outfile** <outfile>

-m, **--matches** <matches>
Required

map

Execute a mapping file and emit objects

```
ftm map [OPTIONS] MAPPING_YAML
```

Options

-o, **--outfile** <outfile>

--sign, **--no-sign**
Apply HMAC signature

Arguments

MAPPING_YAML
Required argument

map-csv

Map CSV data from stdin and emit objects

```
ftm map-csv [OPTIONS] MAPPING_YAML
```

Options

-i, **--infile** <infile>

-o, **--outfile** <outfile>

--sign, **--no-sign**
Apply HMAC signature

Arguments

MAPPING_YAML
Required argument

match-decide

Generate match decisions based purely on score

```
ftm match-decide [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-t, --threshold <threshold>

match-entities

Unnests matches into entities

```
ftm match-entities [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-a, --all
Unnest non-positive matches

pretty

Format a stream of entities to make it readable

```
ftm pretty [OPTIONS]
```

Options

-i, --infile <infile>

sieve

Filter out parts of entities.

```
ftm sieve [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-s, --schema <schema>
Filter out the given schemata.

Options Document | Workbook | PlainText | Table | Package | Family | ProjectParticipant | Company | LegalEntity | Assessment | CourtCase | Person | Article | Pages | Debt | Folder | Interest | BankAccount | Message | License | Page | Sanction | Interval | Event | Documentation | Organization | CryptoWallet | Airplane | Image | Representation | Vessel | Employment | Asset | Thing | Vehicle | Succession | Value | Payment | Identification | Analyzable | CourtCaseParty | Security | Passport | RealEstate | Project | Video | Ownership | Audio | Call | ContractAward | UnknownLink | EconomicActivity | Address | Contract | TaxRoll | Mention | PublicBody | Email | Associate | HyperText | Directorship | Membership | UserAccount | Note

-p, --property <property>
Filter out the given property names.

-t, --type <type>
Filter out the given property types.

Options iban | number | country | date | ip | text | address | name | mimetype | url | json | string | entity | phone | language | identifier | domain | html | checksum | topic | email

sign

Apply a HMAC signature to entity IDs

```
ftm sign [OPTIONS]
```

Options

-i, --infile <infile>

-o, --outfile <outfile>

-s, --signature <signature>
HMAC signature key

validate

Re-parse and validate the given data

```
ftm validate [OPTIONS]
```

Options

-i, **--infile** <infile>

-o, **--outfile** <outfile>

CHAPTER

EIGHT

NETWORK GRAPHS

TODO: Import the graph tooling documentation into this documentation:

https://docs.alephdata.org/developers/followthemoney/ftm#exporting-data-to-a-network-graph

8.1 Implementation

class followthemoney.graph.**Graph**(*edge_types={<iban>, <identifier>, <url>, <ip>, <phone>, <checksum>, <entity>, <address>, <name>, <email>}*)

A set of nodes and edges, derived from entities and their properties. This represents an alternative interpretation of FtM data as a property graph.

This class is meant to be extensible in order to support additional backends, like Aleph.

add(*proxy*)

Add an *EntityProxy* to the graph and make it either a *Node* or an *Edge*.

flush()

Remove all nodes, edges and proxies from the graph.

get_adjacent(*node*, *prop=None*)

Get all adjacent edges of the given node.

get_inbound(*node*, *prop=None*)

Get all edges pointed at the given node.

get_outbound(*node*, *prop=None*)

Get all edges pointed out from the given node.

iteredges()

Iterate all *edges* in the graph.

iternodes()

Iterate all *nodes* in the graph.

queue(*id_*, *proxy=None*)

Register a reference to an entity in the graph.

property queued

Return a list of all the entities which are referenced from the graph but that haven't been loaded yet. This can be used to get a list of entities that should be included to expand the whole graph by one degree.

to_dict()

Return a dictionary with the graph nodes and edges.

class followthemoney.graph.**Node**(*type_*, *value*, *proxy=None*, *schema=None*)
A node represents either an entity that can be rendered as a node in a graph, or as a re-ified value, like a name, email address or phone number.

property caption
A user-facing label for the current node.

classmethod from_proxy(*proxy*)
For a given *EntityProxy*, return a node based on the entity.

id

property is_entity
Check to see if the node represents an entity. If this is false, the node represents a non-entity property value that has been reified, like a phone number or a name.

proxy

schema

to_dict()
Return a simple dictionary to reflect this graph node.

type

value

class followthemoney.graph.**Edge**(*graph*, *source*, *target*, *proxy=None*, *prop=None*, *value=None*)
A link between two nodes.

graph

id

prop

proxy

schema

property source
The graph node from which the edge originates.

source_id

property source_prop
Get the entity property originating this edge.

property target
The graph node to which the edge points.

target_id

property target_prop
Get the entity property originating this edge.

to_dict()

property type_name
Return a machine-readable descripton of the type of the edge. This is either a property name or a schema name.

weight

8.2 How entities are translated to a graph

Below are some notes about the semantics of *followthemoney* (FtM) as a graph, and how it might be converted to a (Neo4J-style) property graph model. This is how we've been thinking about it in the past, and we can change it, but it would almost certainly require adaptation of the FtM model and the complete re-generation of all Aleph data from scratch to do cleanly.

1. The normal case for what a link is in FtM is a property value (see *References*). When turning this into a property graph model, both the *Passport* and the *Person* become nodes, and the *holder* property becomes an edge with no attributes attached to it.
2. Sometimes we want to talk about the inverse of one of these edges. That's why *holder* has an inverse, *passports* that is added to the *Person* schema. This is mostly used to store the labels that should be used to talk about the passports linked to a person. But it's a `stub` that cannot be written.
3. The weakness of this model is that edges don't have properties, whereas relationships in society always have metadata :) Many of them are limited in time (hence *Interval*). So we added a work-around that allows some schema to sort of contract upon generating a property graph and turn into an edge, rather than a node. For example, *Membership*, *Ownership*. But it's important to keep considering this a *special shortcut*, not the normal case for edges in FtM.
4. Because this is a work-around, it leaves the *stub* properties in a bit of an awkward place: it's not clear conceptually if they refer to the original edge (e.g. a link between a *Person.directorshipsDirector -> Directorship.director*) or the contracted edge (*Person -> directorOf -> Organization*)... It's not been a problem in practice as we get the required metadata from the edge annotation of the contracted schema, but it's a sort of weird "place".

CHAPTER

NINE

SCHEMATA DEFINITIONS

Below is an overview of the default data model shipped with *followthemoney* (FtM). It currently includes 64 schemata that can be selected to generate entities.

Schemata

- *File*
- *Workbook*
- *Text file*
- *Table*
- *Package*
- *Family*
- *Project participant*
- *Company*
- *Legal entity*
- *Assessment*
- *Court case*
- *Person*
- *Article*
- *Document*
- *Debt*
- *Folder*
- *Interest*
- *Bank account*
- *Message*
- *License*
- *Page*
- *Sanction*
- *Interval*

- *Event*
- *Documentation*
- *Organization*
- *Cryptocurrency wallet*
- *Airplane*
- *Image*
- *Representation*
- *Vessel*
- *Employment*
- *Asset*
- *Thing*
- *Vehicle*
- *Succession*
- *Value*
- *Payment*
- *Identification*
- *Analyzable*
- *Case party*
- *Security*
- *Passport*
- *Real estate*
- *Project*
- *Video*
- *Ownership*
- *Audio*
- *Call*
- *Contract award*
- *Other link*
- *Customs declaration*
- *Address*
- *Contract*
- *Tax roll*
- *Mention*
- *Public body*
- *E-Mail*

- *Associate*
- *Web page*
- *Directorship*
- *Membership*
- *User account*
- *Note*

9.1 File

Document

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Thing, *Analyzable*

see *followthemoney.schema.Schema.extends*

descendants:
Workbook, *Message*, *Text file*, *Video*, *Audio*, *Package*, *Table*, *Document*, *Article*, *E-Mail*, *Folder*, *Web page*, *Image*

see *followthemoney.schema.Schema.descendants*

Properties:

contentHash(*checksum*)
Checksum (*Checksum*)

SHA1 hash of the data

title(*string*)
Title (*Label*)

author(*string*)
Author (*Label*)

The original author, not the uploader

generator(*string*)
Generator (*Label*)

The program used to generate this file

crawler(*string*)
Crawler (*Label*)

The crawler used to acquire this file

fileSize(*number*)
File size (*Number*)

fileName(*string*)
File name (*Label*)

extension(*string*)
File extension (*Label*)

encoding(*string*)
File encoding (*Label*)

bodyText(*text*)
Text (*Text*)

messageId(*string*)
Message ID (*Label*)

Message ID of a document; unique in most cases

mimeType(*mimetype*)
MIME type (*MIME-Type*)

language(*language*)
Language (*Language*)

translatedLanguage(*language*)
The language of the translated text (*Language*)

translatedText(*text*)
Translated version of the body text (*Text*)

date(*date*)
Date (*Date*)

If not otherwise specified

authoredAt(*date*)
Authored on (*Date*)

publishedAt(*date*)
Published on (*Date*)

parent(*entity*)
Folder (*Entity*, of type *Folder*)

ancestors(*entity*)
Ancestors (*Entity*, of type *Folder*)

processingStatus(*string*)
Processing status (*Label*)

processingError(*string*)
Processing error (*Label*)

And all properties from *Thing*, *Analyzable*.

Properties used as caption:
`fileName`, `title`

see *followthemoney.schema.Schema.caption*

Important properties:
`title`, `fileName`, `mimeType`, `parent`

see *followthemoney.schema.Schema.featured*

9.2 Workbook

Workbook
A spreadsheet document, for example from Excel. Each spreadsheet contains a set of sheets that hold actual data.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Folder

see *followthemoney.schema.Schema.extends*

Properties:
And all properties from *Folder*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, parent

see *followthemoney.schema.Schema.featured*

9.3 Text file

PlainText
Text files, like .txt or source code.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

descendants:
Message, *E-Mail*

see *followthemoney.schema.Schema.descendants*

Properties:
And all properties from *File*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.4 Table

Table
A document structured into rows and cells. This includes simple CSV files, spreadsheet sheets or database relations.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:

columns(*json*)
Column headings (*Nested data*)

rowCount(*number*)
Number of rows (*Number*)

csvHash(*checksum*)
CSV alternative version checksum (*Checksum*)

And all properties from *File*.

Properties used as caption:
title, name, fileName

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.5 Package

Package
A bundle of files that have been packaged together into some form of archive.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Folder

see *followthemoney.schema.Schema.extends*

Properties:
And all properties from *Folder*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
`title`, `fileName`, `mimeType`, `parent`

see *followthemoney.schema.Schema.featured*

9.6 Family

Family
Family relationship between two people

edge: `relative`
In a network graph, this schema is converted into an edge between `person` and `relative`.

see *followthemoney.schema.Schema.edge*

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

person(*entity*)
Person (*Entity*, of type *Person*)

The subject of the familial relation.

relative(*entity*)
Relative (*Entity*, of type *Person*)

The relative of the subject person.

relationship(*string*)
Relationship (*Label*)

Nature of the relationship, from the person's perspective eg. 'mother', where 'relative' is mother of 'person'.

And all properties from *Interval*.

Important properties:
`person`, `relative`, `relationship`

see *followthemoney.schema.Schema.featured*

9.7 Project participant

ProjectParticipant
An activity carried out by a group of participants.

edge: `project`
In a network graph, this schema is converted into an edge between `participant` and `project`.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

project(*entity*)
Project (*Entity*, of type *Project*)

participant(*entity*)
Participant (*Entity*, of type *Legal entity*)

And all properties from *Interest*.

Properties used as caption:
`role`

see *followthemoney.schema.Schema.caption*

Important properties:
`project`, `participant`, `role`

see *followthemoney.schema.Schema.featured*

9.8 Company

Company
A corporation, usually for profit. Does not distinguish between private and public companies, and can also be used to model more specific constructs like trusts and funds. Companies are assets, so they can be owned by other legal entities.

extends:
Organization, *Asset*

see *followthemoney.schema.Schema.extends*

Properties:

jurisdiction(*country*)
Jurisdiction (*Country*)

registrationNumber(*identifier*)
Registration number (*Identifier*)

capital(*string*)
Capital (*Label*)

voenCode(*identifier*)
VOEN (*Identifier*)

Azerbaijan taxpayer ID

coatoCode(*identifier*)
COATO / SOATO / OKATO (*Identifier*)

irsCode(*identifier*)
IRS Number (*Identifier*)

US tax ID

ipoCode(*identifier*)
IPO (*Identifier*)

cikCode(*identifier*)
SEC Central Index Key (*Identifier*)

US SEC Central Index Key

jibCode(*identifier*)
JIB (*Identifier*)

Yugoslavia company ID

mbsCode(*identifier*)
MBS (*Identifier*)

ibcRuc(*identifier*)
ibcRUC (*Identifier*)

caemCode(*string*)
COD CAEM (*Label*)

(RO) What kind of activity a legal entity is allowed to develop

kppCode(*identifier*)
KPP (*Identifier*)

(RU,) in addition to INN for orgs; reason for registration at FNS

okvedCode(*string*)
OKVED(2) Classifier (*Label*)

(RU,) Economical activity classifier. OKVED2 is the same but newer

okopfCode(*string*)
OKOPF (*Label*)

(RU,) What kind of business entity

fnsCode(*identifier*)
Federal tax service code (*Identifier*)

(RU,) Federal Tax Service related info

fssCode(*string*)
FSS (*Label*)

(RU,) Social Security

ogrnCode(*identifier*)
OGRN (*Identifier*)

Major State Registration Number

bikCode(*string*)
BIK (*Label*)

Russian bank account code

pfrNumber(*identifier*)
PFR Number (*Identifier*)

(RU,) Pension Fund Registration number. AAA-BBB-CCCCCC, where AAA is organisation region, BBB is district, CCCCCC number at a specific branch

oksmCode(*string*)
OKSM (*Label*)

Russian () countries classifer

And all properties from *Organization*, *Asset*.

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
name, jurisdiction, registrationNumber, incorporationDate

see *followthemoney.schema.Schema.featured*

9.9 Legal entity

LegalEntity
Any party to legal proceedings, such as asset ownership, corporate governance or social interactions. Often used when raw data does not specify if something is a person or company.

extends:
Thing

see *followthemoney.schema.Schema.extends*

descendants:
Organization, *Company*, *Public body*, *Person*

see *followthemoney.schema.Schema.descendants*

Properties:

name(*name*)
Name (*Name*)

email(*email*)
E-Mail (*E-Mail Address*)

Email address

phone(*phone*)
Phone (*Phone number*)

Phone number

website(*url*)
Website (*URL*)

Website address

legalForm(*string*)
Legal form (*Label*)

incorporationDate(*date*)
Incorporation date (*Date*)

The date the legal entity was incorporated

dissolutionDate(*date*)
Dissolution date (*Date*)

The date the legal entity was dissolved, if applicable

taxStatus(*string*)
Tax status (*Label*)

status(*string*)
Status (*Label*)

sector(*string*)
Sector (*Label*)

classification(*string*)
Classification (*Label*)

registrationNumber(*identifier*)
Registration number (*Identifier*)

Company registration number

idNumber(*identifier*)
ID Number (*Identifier*)

ID number of any applicable ID

taxNumber(*identifier*)
Tax Number (*Identifier*)

Tax identification number

vatCode(*identifier*)
V.A.T. Identifier (*Identifier*)

(EU) VAT number

jurisdiction(*country*)
Jurisdiction (*Country*)

Country or region in which this entity operates

mainCountry(*country*)
Country of origin (*Country*)

Primary country of this entity

opencorporatesUrl(*url*)
OpenCorporates URL (*URL*)

bvdId(*identifier*)
Bureau van Dijk ID (*Identifier*)

icijId(*string*)
ICIJ ID (*Label*)

ID according to International Consortium for Investigative Journalists

okpoCode(*identifier*)
OKPO (*Identifier*)

Russian industry classifier

innCode(*identifier*)
INN (*Identifier*)

Russian company ID

dunsCode(*identifier*)
D-U-N-S (*Identifier*)

Dun & Bradstreet identifier

swiftBic(*identifier*)
SWIFT/BIC (*Identifier*)

Bank identifier code

parent(*entity*)
Parent company (*Entity*, of type *Legal entity*)

If this entity is a subsidiary, another entity (company or organisation) is its parent

And all properties from *Thing*.

Properties used as caption:
name, email, phone, registrationNumber

see *followthemoney.schema.Schema.caption*

Important properties:
name, country, legalForm, status

see *followthemoney.schema.Schema.featured*

9.10 Assessment

Assessment

extends:
Thing

see *followthemoney.schema.Schema.extends*

Properties:

publishDate(*date*)
Date of publishing (*Date*)

assessmentId(*string*)
Assessment ID (*Label*)

author(*entity*)
Author (*Entity*, of type *Legal entity*)

And all properties from *Thing*.

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
name, publishDate, author

see *followthemoney.schema.Schema.featured*

9.11 Court case

CourtCase

extends:
Thing

see *followthemoney.schema.Schema.extends*

Properties:

category(*string*)
Category (*Label*)

type(*string*)
Type (*Label*)

status(*string*)
Status (*Label*)

caseNumber(*identifier*)
Case number (*Identifier*)

court(*string*)
Court (*Label*)

fileDate(*date*)
File date (*Date*)

closeDate(*date*)
Close date (*Date*)

And all properties from *Thing*.

Properties used as caption:
name, caseNumber

see *followthemoney.schema.Schema.caption*

Important properties:
name, fileDate, caseNumber

see *followthemoney.schema.Schema.featured*

9.12 Person

Person
A natural person, as opposed to a corporation of some type.

extends:
Legal entity

see *followthemoney.schema.Schema.extends*

Properties:

title(*string*)
Title (*Label*)

firstName(*string*)
First name (*Label*)

secondName(*string*)
Second name (*Label*)

middleName(*string*)
Middle name (*Label*)

fatherName(*string*)
Patronymic (*Label*)

motherName(*string*)
Matronymic (*Label*)

lastName(*string*)
Last name (*Label*)

birthDate(*date*)
Birth date (*Date*)

birthPlace(*string*)
Place of birth (*Label*)

deathDate(*date*)
Death date (*Date*)

position(*string*)
Position (*Label*)

nationality(*country*)
Nationality (*Country*)

passportNumber(*identifier*)
Passport number (*Identifier*)

gender(*string*)
Gender (*Label*)

ethnicity(*string*)
Ethnicity (*Label*)

religion(*string*)
Religion (*Label*)

political(*string*)
Political association (*Label*)

education(*string*)
Education (*Label*)

And all properties from *Legal entity*.

Properties used as caption:
name, lastName, email, phone

see *followthemoney.schema.Schema.caption*

Important properties:
name, nationality, birthDate

see *followthemoney.schema.Schema.featured*

9.13 Article

Article
A piece of media reporting about a subject.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:
And all properties from *File*.

Properties used as caption:
title, fileName

see *followthemoney.schema.Schema.caption*

Important properties:
title, author, publishedAt

see *followthemoney.schema.Schema.featured*

9.14 Document

Pages
A multi-page document, such as a PDF or Word file or slide-show presentation.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:

pdfHash(*checksum*)
PDF alternative version checksum (*Checksum*)

And all properties from *File*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.15 Debt

Debt

A monetary debt between two parties.

edge: `creditor`
In a network graph, this schema is converted into an edge between `debtor` and `creditor`.

see *followthemoney.schema.Schema.edge*

extends:
Value, *Interval*

see *followthemoney.schema.Schema.extends*

Properties:

debtor(*entity*)
Debtor (*Entity*, of type *Legal entity*)

creditor(*entity*)
creditor (*Entity*, of type *Legal entity*)

And all properties from *Value*, *Interval*.

Important `properties:`
`debtor`, `creditor`, `date`, `amount`

see *followthemoney.schema.Schema.featured*

9.16 Folder

Folder

generated: `True`
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

descendants:
Workbook, *Message*, *E-Mail*, *Package*

see *followthemoney.schema.Schema.descendants*

Properties:
And all properties from *File*.

Properties `used as caption:`
`fileName`, `title`

see *followthemoney.schema.Schema.caption*

Important `properties:`
`title`, `parent`

see *followthemoney.schema.Schema.featured*

9.17 Interest

Interest

abstract: True
see *followthemoney.schema.Schema.abstract*

extends:
Interval

see *followthemoney.schema.Schema.extends*

descendants:
Succession, *Project participant*, *Representation*, *Case party*, *Membership*, *Other link*, *Documentation*, *Ownership*, *Employment*, *Contract award*, *Directorship*

see *followthemoney.schema.Schema.descendants*

Properties:

role(*string*)
Role (*Label*)

status(*string*)
Status (*Label*)

And all properties from *Interval*.

9.18 Bank account

BankAccount
An account held at a bank and controlled by an owner. This may also be used to describe more complex arrangements like correspondent bank settlement accounts.

extends:
Asset

see *followthemoney.schema.Schema.extends*

Properties:

bankName(*string*)
Bank name (*Label*)

accountNumber(*identifier*)
Account number (*Identifier*)

iban(*iban*)
IBAN (*IBAN*)

bic(*identifier*)
Bank Identifier Code (*Identifier*)

bank(*entity*)
Bank (*Entity*, of type *Organization*)

accountType(*string*)
Account type (*Label*)

balance(*number*)
Balance (*Number*)

balanceDate(*date*)
Balance date (*Date*)

maxBalance(*number*)
Maximum balance (*Number*)

maxBalanceDate(*date*)
Maximum balance date (*Date*)

bankAddress(*string*)
Bank address (*Label*)

And all properties from *Asset*.

Properties used as caption:
name, iban, accountNumber

see *followthemoney.schema.Schema.caption*

Important properties:
accountNumber, bankName

see *followthemoney.schema.Schema.featured*

9.19 Message

Message

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Interval, *Text file*, *Web page*, *Folder*

see *followthemoney.schema.Schema.extends*

Properties:

subject(*string*)
Subject (*Label*)

threadTopic(*string*)
Thread topic (*Label*)

sender(*entity*)
Sender (*Entity*, of type *Legal entity*)

senderAccount(*entity*)
Sender Account (*Entity*, of type *User account*)

recipients(*entity*)
Recipients (*Entity*, of type *Legal entity*)

recipientAccount(*entity*)
Recipient Account (*Entity*, of type *User account*)

inReplyTo(*string*)
In Reply To (*Label*)

Message ID of the preceding message in the thread

inReplyToMessage(*entity*)
Responding to (*Entity*, of type *Message*)

metadata(*json*)
Metadata (*Nested data*)

And all properties from *Interval*, *Text file*, *Web page*, *Folder*.

Properties used as caption:
`subject`, `title`, `threadTopic`, `fileName`

see *followthemoney.schema.Schema.caption*

Important properties:
`subject`, `date`, `sender`, `recipients`

see *followthemoney.schema.Schema.featured*

9.20 License

License
A grant of land, rights or property. A type of Contract

extends:
Contract

see *followthemoney.schema.Schema.extends*

Properties:

area(*string*)
Area (*Label*)

commodities(*string*)
Commodities (*Label*)

reviewDate(*string*)
License review date (*Label*)

And all properties from *Contract*.

Properties used as caption:
`name`

see *followthemoney.schema.Schema.caption*

Important properties:
`name`, `amount`, `authority`, `contractDate`, `commodities`

see *followthemoney.schema.Schema.featured*

9.21 Page

Page

hidden: True
see *followthemoney.schema.Schema.hidden*

generated: True
see *followthemoney.schema.Schema.generated*

Properties:

index(*number*)
Index (*Number*)

bodyText(*text*)
Text (*Text*)

document(*entity*)
Document (*Entity*, of type *Document*)

detectedLanguage(*language*)
Detected language (*Language*)

translatedText(*text*)
Translated version of the body text (*Text*)

translatedTextLanguage(*string*)
The language of the translated text (*Label*)

indexText(*text*)
Index text (*Text*)

9.22 Sanction

Sanction
A sanction designation

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

entity(*entity*)
Entity (*Entity*, of type *Thing*)

authority(*string*)
Authority (*Label*)

program(*string*)
Program (*Label*)

status(*string*)
Status (*Label*)

duration(*string*)
Duration (*Label*)

reason(*text*)
Reason (*Text*)

country(*country*)
Country (*Country*)

listingDate(*date*)
Listing date (*Date*)

And all properties from *Interval*.

Properties used as caption:
`program`

see *followthemoney.schema.Schema.caption*

Important properties:
`entity`, `authority`, `program`, `startDate`, `endDate`

see *followthemoney.schema.Schema.featured*

9.23 Interval

Interval
An object which is bounded in time.

abstract: `True`
see *followthemoney.schema.Schema.abstract*

descendants:
Contract award, *Call*, *Tax roll*, *Representation*, *Membership*, *Other link*, *Customs declaration*, *Payment*, *Debt*, *Passport*, *Identification*, *Family*, *Interest*, *Message*, *Event*, *Sanction*, *Associate*, *Address*, *Project participant*, *Succession*, *Case party*, *Project*, *Documentation*, *Ownership*, *Employment*, *Directorship*

see *followthemoney.schema.Schema.descendants*

Properties:

startDate(*date*)
Start date (*Date*)

endDate(*date*)
End date (*Date*)

date(*date*)
Date (*Date*)

summary(*text*)
Summary (*Text*)

description(*text*)
Description (*Text*)

recordId(*string*)
Record ID (*Label*)

sourceUrl(*url*)
Source link (*URL*)

publisher(*string*)
Publishing source (*Label*)

publisherUrl(*url*)
Publishing source URL (*URL*)

alephUrl(*url*)
Aleph URL (*URL*)

namesMentioned(*name*)
Detected names (*Name*)

indexText(*text*)
Index text (*Text*)

modifiedAt(*date*)
Modified on (*Date*)

retrievedAt(*date*)
Retrieved on (*Date*)

9.24 Event

Event

extends:
Thing, *Interval*, *Analyzable*

see *followthemoney.schema.Schema.extends*

Properties:

location(*address*)
Location (*Address*)

country(*country*)
Country (*Country*)

important(*string*)
Important (*Label*)

organizer(*entity*)
Organizer (*Entity*, of type *Legal entity*)

involved(*entity*)
Involved (*Entity*, of type *Legal entity*)

And all properties from *Thing*, *Interval*, *Analyzable*.

Properties used as caption:
name, summary, date

see *followthemoney.schema.Schema.caption*

Important properties:
name, summary, date, location

see *followthemoney.schema.Schema.featured*

9.25 Documentation

Documentation
Links some entity to a document, which might provide further detail or evidence regarding the entity.

edge: `entity`
In a network graph, this schema is converted into an edge between `document` and `entity`.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

document(*entity*)
Document (*Entity*, of type *File*)

entity(*entity*)
Entity (*Entity*, of type *Thing*)

And all properties from *Interest*.

Important `properties:`
`document`, `entity`, `role`

see *followthemoney.schema.Schema.featured*

9.26 Organization

Organization
Any type of incorporated entity that cannot be owned by another (see Company). This might include charities, foundations or state-owned enterprises, depending on their juristiction.

extends:
Legal entity

see *followthemoney.schema.Schema.extends*

descendants:
Company, *Public body*

see *followthemoney.schema.Schema.descendants*

Properties:
And all properties from *Legal entity*.

Properties `used as caption:`
`name`

see *followthemoney.schema.Schema.caption*

Important `properties:`
`name`, `country`, `legalForm`, `status`

see *followthemoney.schema.Schema.featured*

9.27 Cryptocurrency wallet

CryptoWallet

A cryptocurrency wallet is a view on the transactions conducted by one participant on a blockchain / distributed ledger system.

extends:

Thing, *Value*

see *followthemoney.schema.Schema.extends*

Properties:

publicKey(*identifier*)
Address (*Identifier*)

Public key used to identify the wallet

privateKey(*string*)
Private key (*Label*)

creationDate(*date*)
Creation date (*Date*)

currencySymbol(*string*)
Currency short code (*Label*)

mangingExchange(*string*)
Managing exchange (*Label*)

holder(*entity*)
Wallet holder (*Entity*, of type *Legal entity*)

balance(*number*)
Balance (*Number*)

balanceDate(*date*)
Balance date (*Date*)

And all properties from *Thing*, *Value*.

Properties used as caption:
publicKey, name, summary

see *followthemoney.schema.Schema.caption*

Important properties:
currency, publicKey

see *followthemoney.schema.Schema.featured*

9.28 Airplane

Airplane
An airplane, helicopter or other flying vehicle.

extends:
Vehicle

see *followthemoney.schema.Schema.extends*

Properties:

serialNumber(*identifier*)
Serial Number (*Identifier*)

icaoCode(*string*)
ICAO aircraft type designator (*Label*)

manufacturer(*string*)
Manufacturer (*Label*)

And all properties from *Vehicle*.

Properties used as caption:
name, registrationNumber

see *followthemoney.schema.Schema.caption*

Important properties:
type, registrationNumber, country, operator, owner

see *followthemoney.schema.Schema.featured*

9.29 Image

Image
An image file.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:
And all properties from *File*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.30 Representation

Representation
A mediatory, intermediary, middleman, or broker acting on behalf of a legal entity.

edge: `client`
In a network graph, this schema is converted into an edge between `agent` and `client`.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

agent(*entity*)
Agent (*Entity*, of type *Legal entity*)

client(*entity*)
Client (*Entity*, of type *Legal entity*)

And all properties from *Interest*.

Important `properties`:
`agent`, `client`, `role`

see *followthemoney.schema.Schema.featured*

9.31 Vessel

Vessel
A boat or ship. Typically flying some sort of national flag.

extends:
Vehicle

see *followthemoney.schema.Schema.extends*

Properties:

imoNumber(*identifier*)
IMO Number (*Identifier*)

crsNumber(*identifier*)
CRS Number (*Identifier*)

flag(*country*)
Flag (*Country*)

registrationPort(*string*)
Port of Registration (*Label*)

navigationArea(*string*)
Navigation Area (*Label*)

tonnage(*string*)
Tonnage (*Label*)

grossRegisteredTonnage(*number*)
Gross Registered Tonnage (*Number*)

nameChangeDate(*date*)
Date of Name Change (*Date*)

callSign(*identifier*)
Call Sign (*Identifier*)

pastNames(*name*)
Past Names (*Name*)

pastFlags(*string*)
Past Flags (*Label*)

pastTypes(*string*)
Past Types (*Label*)

mmsi(*identifier*)
MMSI (*Identifier*)

And all properties from *Vehicle*.

Properties used as caption:
name, imoNumber

see *followthemoney.schema.Schema.caption*

Important properties:
name, imoNumber, type, flag

see *followthemoney.schema.Schema.featured*

9.32 Employment

Employment

edge: employer
In a network graph, this schema is converted into an edge between employee and employer.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

employer(*entity*)
Employer (*Entity*, of type *Organization*)

employee(*entity*)
Employee (*Entity*, of type *Person*)

And all properties from *Interest*.

Properties used as caption:
role

see *followthemoney.schema.Schema.caption*

Important properties:
employer, employee, role, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.33 Asset

Asset
A piece of property which can be owned and assigned a monetary value.

extends:
Thing, *Value*

see *followthemoney.schema.Schema.extends*

descendants:
Airplane, *Company*, *Vehicle*, *License*, *Vessel*, *Security*, *Contract*, *Real estate*, *Bank account*

see *followthemoney.schema.Schema.descendants*

Properties:
And all properties from *Thing*, *Value*.

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
name, amount

see *followthemoney.schema.Schema.featured*

9.34 Thing

Thing

abstract: True
see *followthemoney.schema.Schema.abstract*

descendants:
Workbook, *Assessment*, *Video*, *Court case*, *Company*, *License*, *Organization*, *Contract*, *Web page*, *Image*, *Airplane*, *Text file*, *Person*, *Note*, *User account*, *Security*, *Bank account*, *Message*, *Public body*, *Audio*, *Cryptocurrency wallet*, *Vehicle*, *Table*, *Document*, *Event*, *Asset*, *Folder*, *Real estate*, *Project*, *Vessel*, *Package*, *File*, *Article*, *E-Mail*, *Legal entity*

see *followthemoney.schema.Schema.descendants*

Properties:

name(*string*)
Name (*Label*)

summary(*text*)
Summary (*Text*)

description(*text*)
Description (*Text*)

country(*country*)
Country (*Country*)

alias(*name*)
Other name (*Name*)

previousName(*name*)
Previous name (*Name*)

weakAlias(*string*)
Weak alias (*Label*)

sourceUrl(*url*)
Source link (*URL*)

publisher(*string*)
Publishing source (*Label*)

publisherUrl(*url*)
Publishing source URL (*URL*)

alephUrl(*url*)
Aleph URL (*URL*)

wikipediaUrl(*url*)
Wikipedia Article (*URL*)

wikidataId(*identifier*)
Wikidata ID (*Identifier*)

keywords(*string*)
Keywords (*Label*)

topics(*topic*)
Topics (*Topic*)

address(*address*)
Address (*Address*)

addressEntity(*entity*)
Address (*Entity*, of type *Address*)

program(*string*)
Program (*Label*)

notes(*text*)
Notes (*Text*)

proof(*entity*)
Source document (*Entity*, of type *File*)

indexText(*text*)
Index text (*Text*)

modifiedAt(*date*)
Modified on (*Date*)

retrievedAt(*date*)
Retrieved on (*Date*)

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
name, country

see *followthemoney.schema.Schema.featured*

9.35 Vehicle

Vehicle

extends:
Asset

see *followthemoney.schema.Schema.extends*

descendants:
Vessel, *Airplane*

see *followthemoney.schema.Schema.descendants*

Properties:

registrationNumber(*identifier*)
Registration number (*Identifier*)

type(*string*)
Type (*Label*)

model(*string*)
Model (*Label*)

owner(*entity*)
Owner (*Entity*, of type *Legal entity*)

operator(*entity*)
Operator (*Entity*, of type *Legal entity*)

buildDate(*date*)
Build Date (*Date*)

registrationDate(*date*)
Registration Date (*Date*)

And all properties from *Asset*.

Properties used as caption:
name, registrationNumber

see *followthemoney.schema.Schema.caption*

Important properties:
type, name, registrationNumber, country, owner

see *followthemoney.schema.Schema.featured*

9.36 Succession

Succession
Two entities that legally succeed each other.

edge: successor
In a network graph, this schema is converted into an edge between predecessor and successor.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

predecessor(*entity*)
Predecessor (*Entity*, of type *Legal entity*)

successor(*entity*)
Successor (*Entity*, of type *Legal entity*)

And all properties from *Interest*.

Important properties:
predecessor, successor, date

see *followthemoney.schema.Schema.featured*

9.37 Value

Value

abstract: True
see *followthemoney.schema.Schema.abstract*

descendants:
Cryptocurrency wallet, *Company*, *Vehicle*, *Airplane*, *Project*, *License*, *Vessel*, *Security*, *Payment*, *Debt*, *Asset*, *Contract award*, *Contract*, *Real estate*, *Bank account*

see *followthemoney.schema.Schema.descendants*

Properties:

amount(*number*)
Amount (*Number*)

currency(*string*)
Currency (*Label*)

amountUsd(*number*)
Amount in USD (*Number*)

amountEur(*number*)
Amount in EUR (*Number*)

9.38 Payment

Payment

A monetary payment between two parties.

edge: beneficiary

In a network graph, this schema is converted into an edge between payer and beneficiary.

see *followthemoney.schema.Schema.edge*

extends:

Value, *Interval*

see *followthemoney.schema.Schema.extends*

Properties:

sequenceNumber(*string*)
Sequence number (*Label*)

transactionNumber(*string*)
Transaction number (*Label*)

purpose(*text*)
Payment purpose (*Text*)

programme(*string*)
Payment programme (*Label*)

Programme name, funding code, category identifier, etc.

payer(*entity*)
Payer (*Entity*, of type *Legal entity*)

payerAccount(*entity*)
Payer bank account (*Entity*, of type *Bank account*)

beneficiary(*entity*)
Beneficiary (*Entity*, of type *Legal entity*)

beneficiaryAccount(*entity*)
Beneficiary bank account (*Entity*, of type *Bank account*)

contract(*entity*)
Contract (*Entity*, of type *Contract*)

project(*entity*)
Project (*Entity*, of type *Project*)

And all properties from *Value*, *Interval*.

Properties used as caption:
amount

see *followthemoney.schema.Schema.caption*

Important properties:
payer, beneficiary, date, amount, purpose

see *followthemoney.schema.Schema.featured*

9.39 Identification

Identification
An form of identification associated with its holder and some issuing country. This can be used for national ID cards, voter enrollments and similar instruments.

extends:
Interval

see *followthemoney.schema.Schema.extends*

descendants:
Passport

see *followthemoney.schema.Schema.descendants*

Properties:

holder(*entity*)
Identification holder (*Entity*, of type *Legal entity*)

type(*string*)
Type (*Label*)

country(*country*)
Country (*Country*)

number(*identifier*)
Passport number (*Identifier*)

authority(*string*)
Authority (*Label*)

And all properties from *Interval*.

Properties used as caption:
number

see *followthemoney.schema.Schema.caption*

Important properties:
number, country, type, holder, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.40 Analyzable

Analyzable
An entity suitable for being processed via named-entity recognition.

abstract: True
see *followthemoney.schema.Schema.abstract*

generated: True
see *followthemoney.schema.Schema.generated*

descendants:
Workbook, *Message*, *Text file*, *Video*, *Audio*, *Note*, *Table*, *File*, *Package*, *Article*, *Document*, *Event*, *E-Mail*, *Folder*, *Web page*, *Image*

see *followthemoney.schema.Schema.descendants*

Properties:

detectedLanguage(*language*)
Detected language (*Language*)

detectedCountry(*country*)
Detected country (*Country*)

namesMentioned(*name*)
Detected names (*Name*)

peopleMentioned(*name*)
Detected people (*Name*)

companiesMentioned(*name*)
Detected companies (*Name*)

ibanMentioned(*iban*)
Detected IBANs (*IBAN*)

ipMentioned(*ip*)
Detected IP addresses (*IP-Address*)

locationMentioned(*address*)
Detected locations (*Address*)

phoneMentioned(*phone*)
Detected phones (*Phone number*)

emailMentioned(*email*)
Detected e-mail addresses (*E-Mail Address*)

9.41 Case party

CourtCaseParty

edge: `case`
In a network graph, this schema is converted into an edge between `party` and `case`.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

party(*entity*)
Party (*Entity*, of type *Thing*)

case(*entity*)
Case (*Entity*, of type *Court case*)

And all properties from *Interest*.

Important properties:
party, case, role

see *followthemoney.schema.Schema.featured*

9.42 Security

Security
A tradeable financial asset.

extends:
Asset

see *followthemoney.schema.Schema.extends*

Properties:

isin(*identifier*)
ISIN (*Identifier*)

International Securities Identification Number

ticker(*identifier*)
Ticker (*Identifier*)

issuer(*entity*)
Issuer (*Entity*, of type *Legal entity*)

issueDate(*date*)
Date issued (*Date*)

type(*string*)
Type (*Label*)

classification(*string*)
Classification (*Label*)

collateral(*string*)
Collateral (*Label*)

And all properties from *Asset*.

Properties used as caption:
name, isin

see *followthemoney.schema.Schema.caption*

Important properties:
isin, name, issuer, country

see *followthemoney.schema.Schema.featured*

9.43 Passport

Passport
An passport held by a person.

extends:
Identification

see *followthemoney.schema.Schema.extends*

Properties:

passportNumber(*identifier*)
Passport number (*Identifier*)

surname(*string*)
Surname (*Label*)

givenName(*string*)
Given name (*Label*)

birthDate(*date*)
Birth date (*Date*)

birthPlace(*string*)
Place of birth (*Label*)

gender(*string*)
Gender (*Label*)

personalNumber(*identifier*)
Personal number (*Identifier*)

And all properties from *Identification*.

Properties used as caption:
passportNumber, number

see *followthemoney.schema.Schema.caption*

Important properties:
passportNumber, country, type, holder, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.44 Real estate

RealEstate
A piece of land or property.

extends:
Asset

see *followthemoney.schema.Schema.extends*

Properties:

latitude(*number*)
Latitude (*Number*)

longitude(*number*)
Longitude (*Number*)

censusBlock(*string*)
Census block (*Label*)

cadastralCode(*identifier*)
Cadastral code (*Identifier*)

area(*number*)
Area (*Number*)

registrationNumber(*identifier*)
Registration number (*Identifier*)

titleNumber(*identifier*)
Title number (*Identifier*)

tenure(*string*)
Tenure (*Label*)

encumbrance(*string*)
Encumbrance (*Label*)

An encumbrance is a right to, interest in, or legal liability on real property that does not prohibit passing title to the property but that diminishes its value.

propertyType(*string*)
Property type (*Label*)

landType(*string*)
Land type (*Label*)

createDate(*date*)
Record date (*Date*)

And all properties from *Asset*.

Properties used as caption:
`name`, `address`, `registrationNumber`

see *followthemoney.schema.Schema.caption*

Important properties:
`registrationNumber`, `address`, `country`

see *followthemoney.schema.Schema.featured*

9.45 Project

Project
An activity carried out by a group of participants.

extends:
Thing, *Interval*, *Value*

see *followthemoney.schema.Schema.extends*

Properties:

projectId(*identifier*)
Project ID (*Identifier*)

status(*string*)
Status (*Label*)

phase(*string*)
Phase (*Label*)

goal(*string*)
Project goal (*Label*)

And all properties from *Thing*, *Interval*, *Value*.

Properties used as caption:
name, projectId

see *followthemoney.schema.Schema.caption*

Important properties:
name, projectId, startDate

see *followthemoney.schema.Schema.featured*

9.46 Video

Video

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:

duration(*number*)
Duration (*Number*)

Duration of the video in ms

And all properties from *File*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.47 Ownership

Ownership

edge: asset
In a network graph, this schema is converted into an edge between owner and asset.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

owner(*entity*)
Owner (*Entity*, of type *Legal entity*)

asset(*entity*)
Asset (*Entity*, of type *Asset*)

percentage(*string*)
Percentage held (*Label*)

sharesCount(*string*)
Number of shares (*Label*)

sharesValue(*string*)
Value of shares (*Label*)

sharesCurrency(*string*)
Currency of shares (*Label*)

sharesType(*string*)
Type of shares (*Label*)

legalBasis(*string*)
Legal basis (*Label*)

ownershipType(*string*)
Type of ownership (*Label*)

And all properties from *Interest*.

Important properties:
owner, asset, percentage, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.48 Audio

Audio

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

Properties:

duration(*number*)
Duration (*Number*)

Duration of the audio in ms

samplingRate(*number*)
Sampling Rate (*Number*)

Sampling rate of the audio in Hz

And all properties from *File*.

Properties used as caption:
fileName, title

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.49 Call

Call

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

caller(*entity*)
Caller (*Entity*, of type *Legal entity*)

callerNumber(*phone*)
Caller's Number (*Phone number*)

receiver(*entity*)
Receiver (*Entity*, of type *Legal entity*)

receiverNumber(*phone*)
Receiver's Number (*Phone number*)

duration(*number*)
Duration (*Number*)

And all properties from *Interval*.

Properties used as caption:
`callerNumber`, `receiverNumber`

see *followthemoney.schema.Schema.caption*

Important properties:
`callerNumber`, `caller`, `receiverNumber`, `receiver`, `date`

see *followthemoney.schema.Schema.featured*

9.50 Contract award

ContractAward
A contract or contract lot as awarded to a supplier.

edge: supplier
In a network graph, this schema is converted into an edge between `contract` and `supplier`.

see *followthemoney.schema.Schema.edge*

extends:
Value, *Interest*

see *followthemoney.schema.Schema.extends*

Properties:

supplier(*entity*)
Supplier (*Entity*, of type *Legal entity*)

The entity the contract was awarded to

contract(*entity*)
Contract (*Entity*, of type *Contract*)

lotNumber(*string*)
Lot number (*Label*)

documentNumber(*string*)
Document number (*Label*)

documentType(*string*)
Document type (*Label*)

decisionReason(*text*)
Decision reason (*Text*)

cpvCode(*string*)
CPV code (*Label*)

Contract Procurement Vocabulary (what type of goods/services, EU)

nutsCode(*string*)
NUTS code (*Label*)

Nomencalture of Territorial Units for Statistics (NUTS)

amended(*string*)
Amended (*Label*)

Was this award amended, modified or updated by a subsequent document?

And all properties from *Value*, *Interest*.

Important properties:
`supplier`, `contract`, `amount`, `lotNumber`, `decisionReason`

see *followthemoney.schema.Schema.featured*

9.51 Other link

UnknownLink

edge: `object`
In a network graph, this schema is converted into an edge between `subject` and `object`.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

subject(*entity*)
Subject (*Entity*, of type *Thing*)

object(*entity*)
Object (*Entity*, of type *Thing*)

And all properties from *Interest*.

Important properties:
`subject`, `object`, `role`

see *followthemoney.schema.Schema.featured*

9.52 Customs declaration

EconomicActivity
A foreign economic activity

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

contract(*entity*)
Contract (*Entity*, of type *Contract*)

ccdNumber(*identifier*)
Customs Cargo Declaration Number (*Identifier*)

ccdValue(*string*)
CCD Value (*Label*)

Declaration Value

directionOfTransportation(*string*)
Direction of transportation (*Label*)

Direction of transportation (import/export)

customsProcedure(*string*)
Customs Procedure (*Label*)

Customs Procedure — type of customs clearance

vedCode(*identifier*)
FEAC Code (*Identifier*)

( ) Foreign Economic Activity Commodity Code

vedCodeDescription(*string*)
FEAC Code description (*Label*)

( ) Foreign Economic Activity Commodity Code description

goodsDescription(*text*)
Description (*Text*)

Description of goods

declarant(*entity*)
Declarant (*Entity*, of type *Legal entity*)

Customs declarant

sender(*entity*)
Sender (*Entity*, of type *Legal entity*)

Origin of the goods

receiver(*entity*)
Receiver (*Entity*, of type *Legal entity*)

Destination of the goods

contractHolder(*entity*)
Contract holder (*Entity*, of type *Legal entity*)

Customs formalities caretaker

invoiceAmount(*string*)
Invoice Value Amount (*Label*)

Invoice Value of goods

customsAmount(*string*)
Customs Value Amount (*Label*)

Customs Value of goods

dollarExchRate(*string*)
USD Exchange Rate (*Label*)

USD Exchange Rate for the activity

tradingCountry(*country*)
Trading Country (*Country*)

Trading Country of the company which transports the goods via Russian border

departureCountry(*country*)
Country of departure (*Country*)

Country out of which the goods are transported

destinationCountry(*country*)
Country of destination (*Country*)

Final destination for the goods

originCountry(*country*)
Country of origin (*Country*)

Country of origin of goods

bankAccount(*entity*)
Bank Account (*Entity*, of type *Bank account*)

Bank account of the contract

bankRub(*entity*)
Rouble bank (*Entity*, of type *Bank account*)

Bank account for payments in roubles

bankForeign(*entity*)
Foreign currency bank (*Entity*, of type *Bank account*)

Bank account for payments in foreign currency

transport(*entity*)
Transport (*Entity*, of type *Vehicle*)

Means of transportation

And all properties from *Interval*.

Properties used as caption:
summary, goodsDescription, ccdNumber

see *followthemoney.schema.Schema.caption*

Important properties:
sender, receiver, contract, goodsDescription, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.53 Address

Address
A location associated with an entity.

generated: `True`
see *followthemoney.schema.Schema.generated*

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

full(*string*)
Full address (*Label*)

remarks(*string*)
Remarks (*Label*)

Handling instructions, like 'care of'.

postOfficeBox(*string*)
PO Box (*Label*)

A mailbox indentifier at the post office

street(*string*)
Street address (*Label*)

street2(*string*)
Street address (ctd.) (*Label*)

city(*string*)
City (*Label*)

City, town, village or other locality

postalCode(*string*)
Postal code (*Label*)

Zip code or postcode.

region(*string*)
Region (*Label*)

Also province or area.

state(*string*)
State (*Label*)

State or federal unit.

latitude(*number*)
Latitude (*Number*)

longitude(*number*)
Longitude (*Number*)

country(*country*)
Country (*Country*)

And all properties from *Interval*.

Properties used as caption:
full, summary, city, remarks

see *followthemoney.schema.Schema.caption*

Important properties:
full, city, street, country

see *followthemoney.schema.Schema.featured*

9.54 Contract

Contract
An contract or contract lot issued by an authority. Multiple lots may be awarded to different suppliers (see ContractAward).

extends:
Asset

see *followthemoney.schema.Schema.extends*

descendants:
License

see *followthemoney.schema.Schema.descendants*

Properties:

title(*string*)
Contract title (*Label*)

authority(*entity*)
Contract authority (*Entity*, of type *Legal entity*)

project(*entity*)
Project (*Entity*, of type *Project*)

type(*string*)
Type (*Label*)

Type of contract. Potentially W (Works), U (Supplies), S (Services).

contractDate(*date*)
Contract date (*Date*)

procedureNumber(*string*)
Procedure number (*Label*)

procedure(*string*)
Contract procedure (*Label*)

noticeId(*string*)
Contract Award Notice ID (*Label*)

numberAwards(*string*)
Number of awards (*Label*)

status(*string*)
Status (*Label*)

method(*string*)
Procurement method (*Label*)

criteria(*string*)
Contract award criteria (*Label*)

classification(*text*)
Classification (*Text*)

cancelled(*string*)
Cancelled? (*Label*)

language(*language*)
Language (*Language*)

And all properties from *Asset*.

Properties used as caption:
`title`, `name`, `procedureNumber`

see *followthemoney.schema.Schema.caption*

Important properties:
`title`, `amount`, `authority`, `contractDate`

see *followthemoney.schema.Schema.featured*

9.55 Tax roll

TaxRoll
A tax declaration of an individual

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

taxee(*entity*)
Taxee (*Entity*, of type *Legal entity*)

country(*country*)
Country (*Country*)

surname(*string*)
Surname (*Label*)

givenName(*string*)
Given name (*Label*)

birthDate(*date*)
Birth date (*Date*)

income(*string*)
Registered income (*Label*)

taxPaid(*string*)
Amount of tax paid (*Label*)

wealth(*string*)
Registered wealth (*Label*)

And all properties from *Interval*.

Important properties:
taxee, date, income, wealth, taxPaid

see *followthemoney.schema.Schema.featured*

9.56 Mention

Mention

hidden: True
see *followthemoney.schema.Schema.hidden*

generated: True
see *followthemoney.schema.Schema.generated*

Properties:

document(*entity*)
Document (*Entity*, of type *File*)

resolved(*entity*)
Entity (*Entity*, of type *Legal entity*)

name(*name*)
Name (*Name*)

detectedSchema(*string*)
Detected entity type (*Label*)

contextCountry(*country*)
Co-occurring countries (*Country*)

contextPhone(*phone*)
Co-occurring phone numbers (*Phone number*)

contextEmail(*email*)
Co-occurring e-mail addresses (*E-Mail Address*)

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
document, name, resolved

see *followthemoney.schema.Schema.featured*

9.57 Public body

PublicBody
A public body, such as a ministry, department or state company.

extends:
Organization

see *followthemoney.schema.Schema.extends*

Properties:
And all properties from *Organization*.

Properties used as caption:
name

see *followthemoney.schema.Schema.caption*

Important properties:
name, country, legalForm, status

see *followthemoney.schema.Schema.featured*

9.58 E-Mail

Email
An internet mail message. The body can be formatted as plain text and/or HTML, and the message may have any number of attachments.

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Text file, *Web page*, *Folder*

see *followthemoney.schema.Schema.extends*

Properties:

subject(*string*)
Subject (*Label*)

threadTopic(*string*)
Thread topic (*Label*)

sender(*string*)
Sender (*Label*)

from(*string*)
From (*Label*)

to(*string*)
To (*Label*)

cc(*string*)
CC (*Label*)

Carbon copy

bcc(*string*)
BCC (*Label*)

Blind carbon copy

emitters(*entity*)
Emitter (*Entity*, of type *Legal entity*)

recipients(*entity*)
Recipients (*Entity*, of type *Legal entity*)

inReplyTo(*string*)
In Reply To (*Label*)

Message ID of the preceding email in the thread

inReplyToEmail(*entity*)
Responding to (*Entity*, of type *E-Mail*)

headers(*json*)
Raw headers (*Nested data*)

And all properties from *Text file*, *Web page*, *Folder*.

Properties used as caption:
`subject`, `threadTopic`, `title`, `name`, `fileName`

see *followthemoney.schema.Schema.caption*

Important properties:
`subject`, `date`, `from`

see *followthemoney.schema.Schema.featured*

9.59 Associate

Associate
Non-family association between two people

edge: associate
In a network graph, this schema is converted into an edge between `person` and `associate`.

see *followthemoney.schema.Schema.edge*

extends:
Interval

see *followthemoney.schema.Schema.extends*

Properties:

person(*entity*)
Person (*Entity*, of type *Person*)

The subject of the association.

associate(*entity*)
Associate (*Entity*, of type *Person*)

An associate of the subject person.

relationship(*string*)
Relationship (*Label*)

Nature of the association

And all properties from *Interval*.

Important properties:
person, associate, relationship

see *followthemoney.schema.Schema.featured*

9.60 Web page

HyperText

generated: True
see *followthemoney.schema.Schema.generated*

extends:
File

see *followthemoney.schema.Schema.extends*

descendants:
Message, *E-Mail*

see *followthemoney.schema.Schema.descendants*

Properties:

bodyHtml(*html*)
HTML (*HTML*)

And all properties from *File*.

Properties used as caption:
title, fileName

see *followthemoney.schema.Schema.caption*

Important properties:
title, fileName, mimeType, parent

see *followthemoney.schema.Schema.featured*

9.61 Directorship

Directorship

edge: organization
In a network graph, this schema is converted into an edge between director and organization.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

director(*entity*)
Director (*Entity*, of type *Legal entity*)

organization(*entity*)
Organization (*Entity*, of type *Organization*)

secretary(*string*)
Secretary (*Label*)

And all properties from *Interest*.

Properties used as caption:
role

see *followthemoney.schema.Schema.caption*

Important properties:
director, organization, role, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.62 Membership

Membership

edge: organization
In a network graph, this schema is converted into an edge between member and organization.

see *followthemoney.schema.Schema.edge*

extends:
Interest

see *followthemoney.schema.Schema.extends*

Properties:

member(*entity*)
Member (*Entity*, of type *Legal entity*)

organization(*entity*)
Organization (*Entity*, of type *Organization*)

And all properties from *Interest*.

Properties used as caption:
role

see *followthemoney.schema.Schema.caption*

Important properties:
member, organization, role, startDate, endDate

see *followthemoney.schema.Schema.featured*

9.63 User account

UserAccount

generated: True
see *followthemoney.schema.Schema.generated*

extends:
Thing

see *followthemoney.schema.Schema.extends*

Properties:

owner(*entity*)
Owner (*Entity*, of type *Legal entity*)

service(*string*)
Service (*Label*)

email(*email*)
E-Mail (*E-Mail Address*)

number(*phone*)
Phone Number (*Phone number*)

username(*string*)
Username (*Label*)

password(*string*)
Password (*Label*)

And all properties from *Thing*.

Properties used as caption:
username, email, service

see *followthemoney.schema.Schema.caption*

Important properties:
username, service, email, owner

see *followthemoney.schema.Schema.featured*

9.64 Note

Note
An annotation that applies to a document or entity.

extends:
Thing, *Analyzable*

see *followthemoney.schema.Schema.extends*

Properties:

entity(*entity*)
Entity (*Entity*, of type *Thing*)

And all properties from *Thing*, *Analyzable*.

Properties used as caption:
description

see *followthemoney.schema.Schema.caption*

Important properties:
description, entity

see *followthemoney.schema.Schema.featured*

CHAPTER

TEN

PROPERTY TYPES

Each property defined on a given schema has one of the data types below. The data types are all string-based, but define the semantics of the property.

Data types

- *Type API*
- *Address*
- *Checksum*
- *Country*
- *Date*
- *Domain*
- *E-Mail Address*
- *Entity*
- *HTML*
- *IBAN*
- *Identifier*
- *IP-Address*
- *Nested data*
- *Language*
- *MIME-Type*
- *Name*
- *Number*
- *Phone number*
- *Label*
- *Text*
- *Topic*
- *URL*

10.1 Type API

class followthemoney.types.common.**PropertyType**

Base class for all property types.

caption(*value: str*) → str

Return a label for the given property value. This is often the same as the value, but for types like countries or languages, it would return the label, while other values like phone numbers can be formatted to be nicer to read.

clean(*text: Any*, ***kwargs*) → Optional[str]

Create a clean version of a value of the type, suitable for storage in an entity proxy.

clean_text(*text: str*, ***kwargs*) → Optional[str]

Specific types can apply their own cleaning routines here (this is called by `clean` after the value has been converted to a string and null values have been filtered).

compare(*left: str*, *right: str*) → float

Comparisons are a float between 0 and 1. They can assume that the given data is cleaned, but not normalised.

compare_safe(*left: Optional[str]*, *right: Optional[str]*) → float

Compare, but support None values on either side of the comparison.

compare_sets(*left: Sequence[str], right: Sequence[str], func: Callable[[Sequence[float]], float] = <built-in function max>*) → float

Compare two sets of values and select the highest-scored result.

country_hint(*value: str*) → Optional[str]

Determine if the given value allows us to infer a country that it may be related to (e.g. using a country prefix on a phone number or IBAN).

group: Optional[str] = None

Groups are used to invert all the properties of an entity that have a given type into a single list before indexing them. This way, in Aleph, you can query for `countries:gb` instead of having to make a set of filters like `properties.jurisdiction:gb OR properties.country:gb OR ...`.

join(*values: Sequence[str]*) → str

Helper function for converting multi-valued FtM data into formats that allow only a single value per field (e.g. CSV). This is not fully reversible and should be used as a last option.

label: Optional[str] = None

A name for this type to be shown to users.

matchable: bool = True

Matchable types allow properties to be compared with each other in order to assess entity similarity. While it makes sense to compare names, countries or phone numbers, the same isn't true for raw JSON blobs or descriptive text snippets.

max_size: Optional[int] = None

Some types have overall size limitations in place in order to avoid generating entities that are very large (upstream ElasticSearch has a 100MB document limit). Once the total size of all properties of this type has exceed the given limit, an entity will refuse to add further values.

name: str = 'any'

A machine-facing, variable safe name for the given type.

node_id(*value: str*) → str

Return an ID suitable to identify this entity as a typed node in a graph representation of some FtM data. It's usually the same as the the RDF form.

node_id_safe(*value: Optional[str]*) → Optional[str]
Wrapper for node_id to handle None values.

pivot: bool = False
Pivot property types are like a stronger form of *matchable* types: they will be used when value-based lookups are used to find commonalities between entities. For example, pivot typed-properties are used to show all the other entities that mention the same phone number, email address or name as the one currently seen by the user.

plural: Optional[str] = None
A plural name for this type which can be used in appropriate places in a user interface.

rdf(*value: str*) → rdflib.term.Identifier
Return an RDF term to represent the given value - either a string literal, or a URI reference.

specificity(*value: Optional[str]*) → float
Return a score for how specific the given value is. This can be used as a weighting factor in entity comparisons in order to rate matching property values by how specific they are. For example: a longer address is considered to be more specific than a short one, a full date more specific than just a year number, etc.

to_dict() → Dict[str, Any]
Return a serialisable description of this data type.

validate(*text: Any*, ***kwargs*) → bool
Returns a boolean to indicate if the given value is a valid instance of the type.

class followthemoney.types.common.**EnumType**(**args*)
Enumerated type properties are used for types which have a defined set of possible values, like languages and countries.

caption(*value*)
Given a code value, return the label that should be shown to a user.

clean_text(*code*, *guess=False*, ***kwargs*)
All code values are cleaned to be lowercase and trailing whitespace is removed.

property names
Return a mapping from property values to their labels in the current locale.

to_dict()
When serialising the model to JSON, include all values.

validate(*code*, ***kwargs*)
Make sure that the given code value is one of the supported set.

class followthemoney.types.registry.**Registry**
This registry keeps the processing helpers for all property types in the system. They are instantiated as singletons when the system is first loaded. The registry can be used to get a type, which can itself then clean, validate or format values of that type.

add(*clazz: Type[*followthemoney.types.common.PropertyType*]*) → None
Add a singleton class.

get(*name: Union[str,* followthemoney.types.common.PropertyType*]*) →
Optional[*followthemoney.types.common.PropertyType*]
For a given property type name, get its type object. This can also be used via getattr, e.g. `registry.phone`.

get_types(*names: List[Union[str,* followthemoney.types.common.PropertyType*]]*) →
List[*followthemoney.types.common.PropertyType*]
Get a list of all type names.

10.2 Address

address: Addresses

A geographic address used to describe a location of a residence or post box. There is no specified order for the sub-parts of an address (e.g. street, city, postal code), and we should consider introducing an Address schema type to retain fidelity in cases where address parts are specified.

group: addresses
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.3 Checksum

checksum: Checksums

Content hashes calculated using SHA1. Checksum references are used by document-typed entities in Aleph to refer to raw data in the archive (e.g. the document from which the entity is extracted).

Unfortunately, this has some security implications: in order to avoid people getting access to documents for which they know the checksum, properties of this type are scrubbed when submitted via the normal API. Checksums can only be defined by uploading a document to be ingested.

group: checksums
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.4 Country

country: Countries

Properties to define countries and territories. This is completely descriptive and needs to deal with data from many origins, so we support a number of unusual and controversial designations (e.g. the Soviet Union, Transnistria, Somaliland, Kosovo).

group: countries
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

Code value	Label
ac	Ascension Island
ad	Andorra
ae	United Arab Emirates
af	Afghanistan

continues on next page

Table 1 – continued from previous page

Code value	Label
ag	Antigua & Barbuda
ai	Anguilla
al	Albania
am	Armenia
ao	Angola
aq	Antarctica
ar	Argentina
as	American Samoa
at	Austria
au	Australia
aw	Aruba
ax	Åland Islands
az	Azerbaijan
az-nk	Nagorno-Karabakh
ba	Bosnia & Herzegovina
bb	Barbados
bd	Bangladesh
be	Belgium
bf	Burkina Faso
bg	Bulgaria
bh	Bahrain
bi	Burundi
bj	Benin
bl	St. Barthélemy
bm	Bermuda
bn	Brunei
bo	Bolivia
bq	Caribbean Netherlands
br	Brazil
bs	Bahamas
bt	Bhutan
bv	Bouvet Island
bw	Botswana
by	Belarus
bz	Belize
ca	Canada
cc	Cocos (Keeling) Islands
cd	Congo - Kinshasa
cf	Central African Republic
cg	Congo - Brazzaville
ch	Switzerland
ci	Côte d'Ivoire
ck	Cook Islands
cl	Chile
cm	Cameroon
cn	China
cn-xz	Tibet
co	Colombia
cp	Clipperton Island

continues on next page

Table 1 – continued from previous page

Code value	Label
cr	Costa Rica
csxx	Serbia and Montenegro
cu	Cuba
cv	Cape Verde
cw	Curaçao
cx	Christmas Island
cy	Cyprus
cy-trnc	Northern Cyprus
cz	Czech Republic
dd	East Germany
de	Germany
dg	Diego Garcia
dj	Djibouti
dk	Denmark
dm	Dominica
do	Dominican Republic
dz	Algeria
ea	Ceuta & Melilla
ec	Ecuador
ee	Estonia
eg	Egypt
eh	Western Sahara
er	Eritrea
es	Spain
et	Ethiopia
eu	European Union
ez	Eurozone
fi	Finland
fj	Fiji
fk	Falkland Islands
fm	Micronesia
fo	Faroe Islands
fr	France
ga	Gabon
gb	United Kingdom
gb-nir	Northern Ireland
gb-sct	Scotland
gb-wls	Wales
gd	Grenada
ge	Georgia
ge-ab	Abkhazia
gf	French Guiana
gg	Guernsey
gg-srk	Sark
gh	Ghana
gi	Gibraltar
gl	Greenland
gm	Gambia
gn	Guinea

continues on next page

Table 1 – continued from previous page

Code value	Label
gp	Guadeloupe
gq	Equatorial Guinea
gr	Greece
gs	South Georgia & South Sandwich Islands
gt	Guatemala
gu	Guam
gw	Guinea-Bissau
gy	Guyana
hk	Hong Kong SAR China
hm	Heard & McDonald Islands
hn	Honduras
hr	Croatia
ht	Haiti
hu	Hungary
ic	Canary Islands
id	Indonesia
ie	Ireland
il	Israel
im	Isle of Man
in	India
io	British Indian Ocean Territory
iq	Iraq
ir	Iran
is	Iceland
it	Italy
je	Jersey
jm	Jamaica
jo	Jordan
jp	Japan
ke	Kenya
kg	Kyrgyzstan
kh	Cambodia
ki	Kiribati
km	Comoros
kn	St. Kitts & Nevis
kp	North Korea
kr	South Korea
kw	Kuwait
ky	Cayman Islands
kz	Kazakhstan
la	Laos
lb	Lebanon
lc	St. Lucia
li	Liechtenstein
lk	Sri Lanka
lr	Liberia
ls	Lesotho
lt	Lithuania
lu	Luxembourg

continues on next page

Table 1 – continued from previous page

Code value	Label
lv	Latvia
ly	Libya
ma	Morocco
mc	Monaco
md	Moldova
md-pmr	Transnistria
me	Montenegro
mf	St. Martin
mg	Madagascar
mh	Marshall Islands
mk	North Macedonia
ml	Mali
mm	Myanmar (Burma)
mn	Mongolia
mo	Macao SAR China
mp	Northern Mariana Islands
mq	Martinique
mr	Mauritania
ms	Montserrat
mt	Malta
mu	Mauritius
mv	Maldives
mw	Malawi
mx	Mexico
my	Malaysia
mz	Mozambique
na	Namibia
nc	New Caledonia
ne	Niger
nf	Norfolk Island
ng	Nigeria
ni	Nicaragua
nl	Netherlands
no	Norway
np	Nepal
nr	Nauru
nu	Niue
nz	New Zealand
om	Oman
pa	Panama
pe	Peru
pf	French Polynesia
pg	Papua New Guinea
ph	Philippines
pk	Pakistan
pl	Poland
pm	St. Pierre & Miquelon
pn	Pitcairn Islands
pr	Puerto Rico

continues on next page

Table 1 – continued from previous page

Code value	Label
ps	Palestinian Territories
pt	Portugal
pw	Palau
py	Paraguay
qa	Qatar
qo	Outlying Oceania
re	Réunion
ro	Romania
rs	Serbia
ru	Russia
rw	Rwanda
sa	Saudi Arabia
sb	Solomon Islands
sc	Seychelles
sd	Sudan
se	Sweden
sg	Singapore
sh	St. Helena
si	Slovenia
sj	Svalbard & Jan Mayen
sk	Slovakia
sl	Sierra Leone
sm	San Marino
sn	Senegal
so	Somalia
so-som	Somaliland
sr	Suriname
ss	South Sudan
st	São Tomé & Príncipe
suhh	Soviet Union
sv	El Salvador
sx	Sint Maarten
sy	Syria
sz	Eswatini
ta	Tristan da Cunha
tc	Turks & Caicos Islands
td	Chad
tf	French Southern Territories
tg	Togo
th	Thailand
tj	Tajikistan
tk	Tokelau
tl	Timor-Leste
tm	Turkmenistan
tn	Tunisia
to	Tonga
tr	Turkey
tt	Trinidad & Tobago
tv	Tuvalu

continues on next page

Table 1 – continued from previous page

Code value	Label
tw	Taiwan
tz	Tanzania
ua	Ukraine
ug	Uganda
um	U.S. Outlying Islands
un	United Nations
us	United States
uy	Uruguay
uz	Uzbekistan
va	Vatican City
vc	St. Vincent & Grenadines
ve	Venezuela
vg	British Virgin Islands
vi	U.S. Virgin Islands
vn	Vietnam
vu	Vanuatu
wf	Wallis & Futuna
ws	Samoa
x-so	South Ossetia
xa	Pseudo-Accents
xb	Pseudo-Bidi
xk	Kosovo
ye	Yemen
yt	Mayotte
yucs	Yugoslavia
za	South Africa
zm	Zambia
zr	Zaire
zw	Zimbabwe
zz	Global

10.5 Date

date: Dates

A date or time stamp. This is based on ISO 8601, but meant to allow for different degrees of precision by specifying a prefix. This means that `2021`, `2021-02`, `2021-02-16`, `2021-02-16T21`, `2021-02-16T21:48` and `2021-02-16T21:48:52` are all valid values, with an implied precision.

The timezone is always expected to be UTC and cannot be specified otherwise. There is no support for calendar weeks (`2021-W7`) and date ranges (`2021-2024`).

group: dates

see *followthemoney.types.common.PropertyType.group*

matchable: True

see *followthemoney.types.common.PropertyType.matchable*

10.6 Domain

domain: Domains
DNS domain names. Not really used and might be thrown out.

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

10.7 E-Mail Address

email: E-Mail Addresses
Internet mail address (e.g. user@example.com). These are notoriously hard to validate, but we use an irresponsibly simple rule and hope for the best.

pattern: ^[^@\s]+@[^@\s]+\.\w+$

group: emails
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.8 Entity

entity: Entities
A reference to another entity via its ID. This is how entities in FtM become a graph: by pointing at each other using *References*.

Entity IDs can either be *namespaced* or *plain*, depending on the context. When setting properties of this type, you can pass in an entity proxy or dict of the entity, the ID will then be extracted and stored.

pattern: ^[0-9a-zA-Z]([0-9a-zA-Z\.\-]*[0-9a-zA-Z])?$

group: entities
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.9 HTML

html: HTMLs
Properties that contain raw hypertext markup (HTML).

User interfaces rendering properties of this type need to take extreme care not to allow attacks such as cross-site scripting. It is recommended to perform server-side sanitisation, or to not render this property at all.

max_size: 31457280
see *followthemoney.types.common.PropertyType.max_size*

10.10 IBAN

iban: IBANs
An international bank account number, as defined in ISO 13616. IBANs are managed by SWIFT used in the European SEPA payment system.

A noteable aspect of IBANs is that they share a country prefix and validation mechanism, but the specific length of an IBAN is dependent on the country code defined in the first two characters: NO8330001234567 and CY21002001950000357001234567 are both valid values.

group: ibans
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.11 Identifier

identifier: Identifiers
Used for registration numbers and other codes assigned by an authority to identify an entity. This might include tax identifiers and statistical codes.

Since identifiers are high-value criteria when comparing two entities, numbers should only be modelled as identifiers if they are long enough to be meaningful. Four- or five-digit industry classifiers create more noise than value.

group: identifiers
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.12 IP-Address

ip: IP-Addresses
Internet protocol addresses. This supports both addresses used by the protocol versions 4 (e.g. `192.168.1.143`) and 6 (e.g. `0:0:0:0:0:ffff:c0a8:18f`).

group: ips
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.13 Nested data

json: None
An encoded JSON object. This is used to store raw HTTP headers for documents and some other edge cases. It's a really bad idea and we should try to get rid of JSON properties.

10.14 Language

language: Languages
A human written language. This list is arbitrarily limited for some weird upstream technical reasons, but we'll happily accept pull requests for additional languages once there is a specific need for them to be supported.

group: languages
see *followthemoney.types.common.PropertyType.group*

Code value	Label
afr	Afrikaans
ara	Arabic
aze	Azerbaijani
bel	Belarusian
bos	Bosnian
bul	Bulgarian
cat	Catalan
ces	Czech
dan	Danish
deu	German
ell	Greek
eng	English
est	Estonian
fas	Persian
fil	Filipino
fin	Finnish
fra	French
heb	Hebrew
hin	Hindi

continues on next page

Table 2 – continued from previous page

Code value	Label
hrv	Croatian
hun	Hungarian
hye	Armenian
ind	Indonesian
isl	Icelandic
ita	Italian
jpn	Japanese
kan	Kannada
kat	Georgian
kaz	Kazakh
kir	Kyrgyz
kor	Korean
lav	Latvian
lit	Lithuanian
ltz	Luxembourgish
mkd	Macedonian
mlt	Maltese
mon	Mongolian
msa	Malay
mya	Burmese
nep	Nepali
nld	Dutch
nor	Norwegian
pol	Polish
por	Portuguese
ron	Romanian
rus	Russian
slk	Slovak
slv	Slovenian
spa	Spanish
sqi	Albanian
srp	Serbian
swa	Swahili
swe	Swedish
tgk	Tajik
tgl	Tagalog
tuk	Turkmen
tur	Turkish
ukr	Ukrainian
urd	Urdu
uzb	Uzbek
zho	Chinese

10.15 MIME-Type

mimetype: MIME-Types
A MIME media type are a specification of a content type on a network. Each MIME type is assinged by IANA and consists of two parts: the type and sub-type. Common examples are: `text/plain`, `application/json` and `application/pdf`.

MIME type properties do not contain parameters as used in HTTP headers, like `charset=UTF-8`.

group: mimetypes
see *followthemoney.types.common.PropertyType.group*

10.16 Name

name: Names
A name used for a person or company. This is assumed to be as complete a name as available - when a first name, family name or patronymic are given separately, these are stored to string-type properties instead.

No validation rules apply, and things having multiple names must be considered a perfectly ordinary case.

group: names
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.17 Number

number: Numbers
A numeric value, like the size of a piece of land, or the value of a contract. Since all property values in FtM are strings, this is also a string and there is no specified format (e.g. `1,000.00` vs. `1.000,00`).

In the future we might want to enable annotations for format, units, or even to introduce a separate property type for monetary values.

10.18 Phone number

phone: Phone numbers
A phone number in E.164 format. This means that phone numbers always include an international country prefix (e.g. `+38760183628`). The cleaning and validation functions for this try to be smart about by accepting a list of countries as an argument in order to add the number prefix.

When adding a property of this type to an entity, any country-type properties defined for the entity are considered for validation. That means that adding a phone number to an entity before adding a country can have a different validation outcome from doing the two operations the other way around. Always define the country first.

group: phones
see *followthemoney.types.common.PropertyType.group*

matchable: True
see *followthemoney.types.common.PropertyType.matchable*

pivot: True
see *followthemoney.types.common.PropertyType.pivot*

10.19 Label

string: Labels
A simple string property with no additional semantics.

10.20 Text

text: Texts
Longer text fragments, such as descriptions or document text. Unlike string properties, it might make sense to treat properties of this type as full-text search material.

max_size: 31457280
see *followthemoney.types.common.PropertyType.max_size*

10.21 Topic

topic: Topics
Topics define a controlled vocabulary of terms applicable to some entities, such as companies and people. They describe categories of journalistic interest which may apply to the given entity, for example if a given person is a criminal or a politician.

Besides the informative value, topics are ultimately supposed to bear fruits in the context of graph-based data analysis, where they would enable queries such as *find all paths between a government procurement award and a politician*.

group: topics
see *followthemoney.types.common.PropertyType.group*

Code value	Label
asset.frozen	Frozen asset
corp.offshore	Offshore
corp.shell	Shell company
crime	Crime
crime.boss	Criminal leadership
crime.cyber	Cybercrime
crime.fin	Financial crime
crime.fraud	Fraud
crime.terror	Terrorism
crime.theft	Theft
crime.traffick	Trafficking
crime.traffick.drug	Drug trafficking
crime.traffick.human	Human trafficking
crime.war	War crimes

continues on next page

Table 3 – continued from previous page

Code value	Label
ctx.poi	Person of interest
ctx.sanctioned	Sanctioned entity
fin	Financial services
fin.adivsor	Financial advisor
fin.bank	Bank
fin.fund	Fund
gov	Government
gov.igo	Intergovernmental organization
gov.muni	Municipal government
gov.national	National government
gov.soe	State-owned enterprise
gov.state	State government
mil	Military
pol.party	Political party
pol.union	Union
rel	Religion
role.acct	Accountant
role.act	Activist
role.civil	Civil servant
role.diplo	Diplomat
role.journo	Journalist
role.judge	Judge
role.lawyer	Lawyer
role.pep	Politician
role.rca	Associate
role.spy	Spy

10.22 URL

url: URLs

A uniform resource locator (URL). This will perform some normalisation on the URL so that it's sure to be using valid encoding/quoting, and to make sure the URL has a schema (e.g. 'http', 'https', ...).

group: urls

see *followthemoney.types.common.PropertyType.group*

matchable: True

see *followthemoney.types.common.PropertyType.matchable*

pivot: True

see *followthemoney.types.common.PropertyType.pivot*

CHAPTER

ELEVEN

FREQUENTLY ASKED QUESTIONS

11.1 Why not a propery graph model?

Until Aleph 2.x, the *followthemoney* (FtM) system was based on edges and nodes, like you would expect if you're coming from a network analysis background. However, using that model forced us to make more and more random, opinionated, modelling decisions: what is the source of an ownership edge - the owner, or the asset? Is an email message a node or a hypergraph edge with many targets? What about a payment, or a customs declaration? Both often have more than two parties involved.

As soon as we started modelling FtM data as entities only, many of these semantics could actually be expressed, rather than implicitly fudged. There's still "funky" places, like familiar relations which would need to be modelled in much more detail to be unambiguous.

In general, property graphs are much more interpretative than we admit: they try to reduce the complexity of a domain into something targeted at answering a specific set of analytical queries. The entity-only model, on the other hand, remains a bit more descriptive.

CHAPTER

TWELVE

CREDITS

- The Popolo data standard developed by an alliance of open data activists is the foundation of *followthemoney*. It provides the origin of key entities, including *Person*, *Organization* and *Membership*.
- Popolo is implemented by EveryPolitician, establishing a practical reference.
- Dr. Amy Guy took an essential role in breaking *followthemoney* out of Aleph and introducing the current entity-based model.

PYTHON MODULE INDEX

f

INDEX

Symbols

A

C

D

E

F

G

H

I

J

K

L

M

N

U

V

W