Properties overview

Definition

Property

Property is the most fundamental and the most versatile relation in CASTEMO knowledge graphs which connects **entities of different kinds**.

Any property has:

1. an **origin**, i.e. the entity to which it is attached,
2. a **property type**, and
3. a **property value**.

Properties serve for modelling:

- **Adjectives, adjectival expressions, possessives, and appositions** (e.g., qualities and inter-entity relations).
- **Adverbs and adverbial expressions** (e.g., time, space, and manner of action).

CASTEMO knowledge graphs give priority to the **data-collection perspective**, and attempt at creating a **general** framework for modelling information in the textual resources. That's why it does not have a comprehensive list of predefined properties but allows coders to fill them during data collection flexibly and dynamically.

> Any entity can have **any number of properties**.

> If an entity has **more qualities of the same kind**, those will be multiple properties with the same property type (e.g. if something is yellow and blue, it will have one property for yellow and the other for blue colour).

In CASTEMO knowledge graphs, a Property is always read with the verb "has".

E.g.:

```
O John's hat  - [has] - C colour - C black
```

```
P Anna - [has] - C son - P Peter
```

There are two kinds of properties, which have the same structure but differ in the context from which they are attached to entities.

In-statement Properties

In-statement Properties are attached to entities **contextually**, from within a Statement - i.e. they are attached to entities, used as **actants** of a Statement (in the case of adjectival expressions and appositions) or **actions** (in the case of adverbial expressions). Thanks to the position of Statements in CASTEMO knowledge graphs, such Properties are included in a Territory hierarchy from which they stem, and usually accompanied with a either an anchor in a full-text document, or a Reference to the Resource which served for compiling a given Statement.

Meta-level Properties

Meta-level Properties (or, in short, **Metaproperties**), are properties attached **globally** rather than contextually to an entity, and represent general knowledge which applies to all uses of the entity independently from context. For instance:
`P Martha Wood - [has] - C sex - C female`

Sometimes, **one level is not enough** to model a property. E.g., if Elizabeth II was Queen of the United Kingdom from 1952 to 2022, it is important to declare both the role and its temporal span, e.g. in the following way:

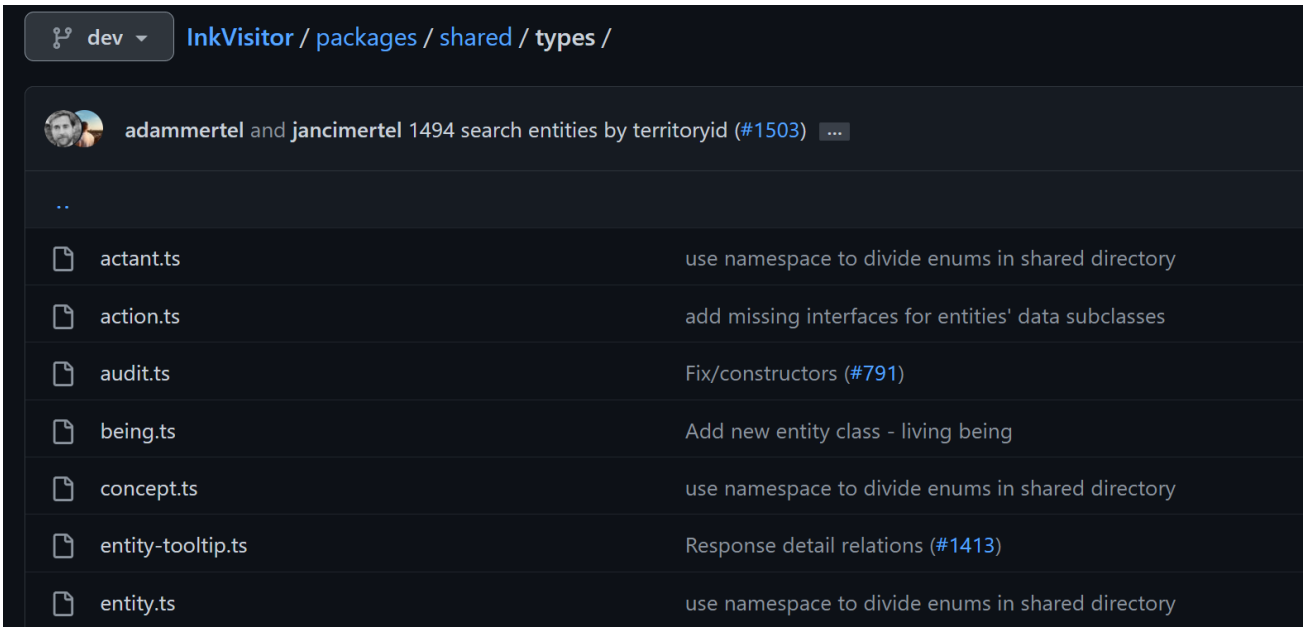
```
P Elizabeth II - [has] - C office - C Queen of the United Kingdom

    PROP - C TRP yyyy-mm-dd: started exactly - V 1952
    PROP - C TRP yyyy-mm-dd: ended exactly - V 2022
```

It is not Elizabeth who has this time span, but her office as queen of the United Kingdom. Thus, CASTEMO knowledge graphs have **second-level properties to be able to attach properties to properties**, and they have even **third-order properties** to attach properties to properties of properties.

Technical specification

The data model of CASTEMO knowledge graphs is **technically** defined in the InkVisitor **typescript *.ts files**, which directly correspond to the JSON structure of the collected data in the corresponding database.

Each **entity** (e.g. Action) has its own **ts file** (e.g. *action.ts*); see the GitHub repository:



In the sense of object-oriented design, each specific entity (e.g. location, person) inherits the definition of the **IEntity interface**, which is in the *entity.ts* file. The **IEntity** is defined as follows:

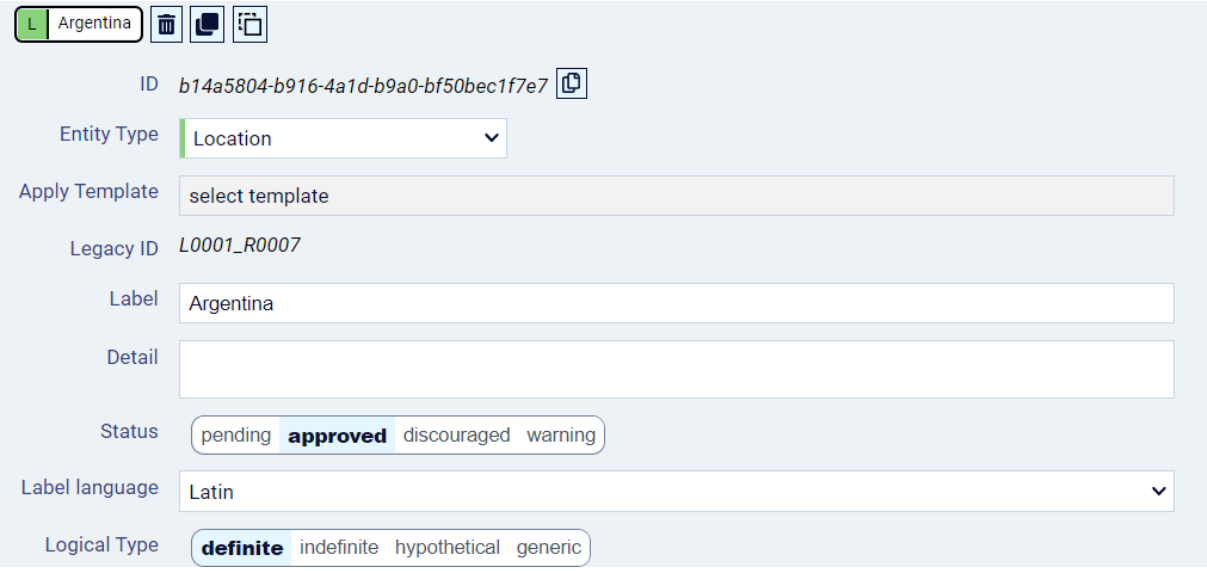
20 lines (19 sloc) | 453 Bytes

```
1  import { IProp } from ".";
2  import { EntityEnums } from "../enums";
3  import { IReference } from "./reference";
4
5  export interface IEntity {
6    id: string;
7    legacyId?: string;
8    class: EntityEnums.Class;
9    status: EntityEnums.Status;
10   data: any;
11   label: string;
12   detail: string;
13   language: EntityEnums.Language;
14   notes: string[];
15   props: IProp[];
16   references: IReference[];
17   isTemplate?: boolean;
18   usedTemplate?: string;
19   templateData?: object;
20 }
```

Example: location entity "Argentina"

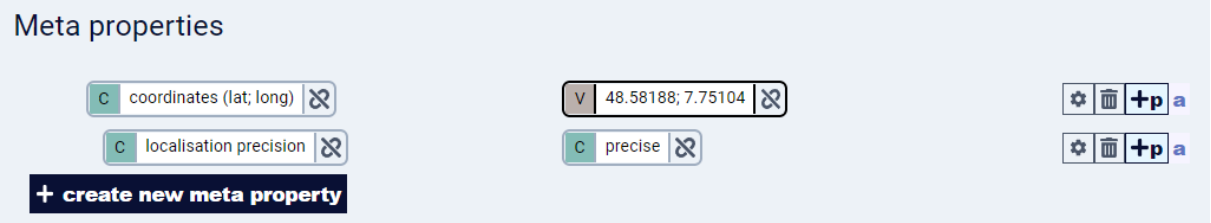
Let's have a location entity with the label "Argentina".

In the **InkVisitor GUI**, the core entity is displayed like this:



...

In the **InkVisitor GUI**, the properties which hold its geospatial localisation are displayed like this:



There is **one property record** with type-value pair, i.e. **Property Type** "coordinates (lat; long)" (a Concept entity, an entity object of type C, which is taken is from the predefined ontology) and the **Property Value**, in this case a **Value object** with the string holding the decimal geographic coordinate pair. To this first-level property is attached a second-level property defining "localisation precision" of the stated coordinates, where both values of the type-value pair are filled with C entities.

For deeper conceptual understanding, the **JSON structure** can be more revealing, this is a full entity display:


```
"root" : {  20 items
  "class" : "L"
  "data" : {...}  1 item
  "detail" : ""
  "entities" : {...}  12 items
  "id" : "b14a5804-b916-4a1d-b9a0-bf50bec1f7e7"
  "label" : "Argentina"
  "language" : "lat"
  "legacyId" : "L0001_R0007"
  "notes" : [...]  1 item
  "props" : [...]  1 item
  "references" : [...]  1 item
  "relations" : {...}  15 items
```

The entity has class "L", i.e. it is an entity of type "location", and it has direct attributes, like *label* with the value "Argentina" and array attributes like *props*.

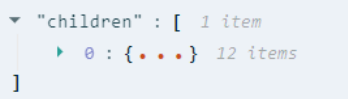
It has one property object in the *props* attribute. The main information is held in **type** and **value** attributes. And this property object also has one **second-level property** in the attribute *children*.

The property object is defined by the class *IProp,* see *prop.ts* file.

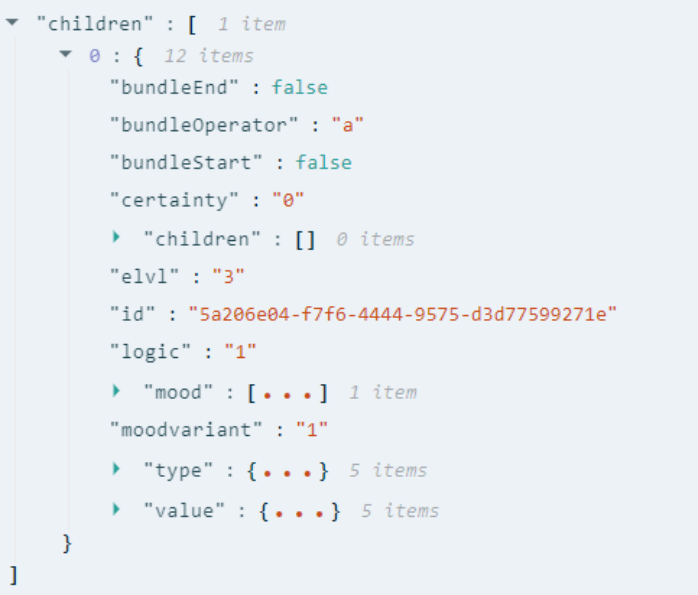
```
export interface IProp {
  id: string;
  elvl: EntityEnums.Elvl;
  certainty: EntityEnums.Certainty;
  logic: EntityEnums.Logic;
  mood: EntityEnums.Mood[];
  moodvariant: EntityEnums.MoodVariant;
  bundleOperator: EntityEnums.Operator;
  bundleStart: boolean;
  bundleEnd: boolean;

  children: IProp[];

  type: IPropSpec;
  value: IPropSpec;
}

export interface IPropSpec {
  entityId: string;
  elvl: EntityEnums.Elvl;
  logic: EntityEnums.Logic;
  virtuality: EntityEnums.Virtuality;
  partitivity: EntityEnums.Partitivity;
}
```

Second-level properties

In the Argentina example above, the coordinates property record has one property object which extends the information.



This object is a normal instance of the IProp class. This means that its primary information is in type and value attributes. These hold *IPropSpec* objects, which refer to two concept entities, e.g. here, "localisation precision" and "precise".



Revision #16
Created 19 December 2022 16:51:22 by Tomáš Hampejs
Updated 23 June 2025 12:28:20 by Prof. Dr. David Zbíral, Ph.D.