On The Nature of Models

Introduces the fundamental principles of CDS models.

Metaphysics of Languages

A *model* is a *thing* that describes *something*. For example, a *data model describes the type structure (commonly also called *'schema'*) of *data*.

Languages

Representations

Models can come in different *representations*, which follow different *syntaxes*. For example, we use the *CDL* syntax for *human-readable* representations of CDS models, while CSN is an *object notation*, i.e. a special form of *syntax*, used for *machine-readable* representations of CDS models.

▶ *On CSN representations...*

Reflections

CDS models can be compiled to other languages, that play in the same fields, yet not covering the same information, but rather with some loss of information — we call these '*reflections*'.

Examples are:

- SQL DDL covers the persistence model interface only → only flat tables and views
- OData EDMX covers the service interfaces only → queryable entities still exist, with implicit features
- GraphQL also covers service interfaces → queryable entities still exist, but without less features
- OpenAPI also covers the service interfaces, with → queryable entities got 'flattened' to paths with input and output types

---

The above principles apply not only to CDS models, but also to Queries:

- CQL is a syntax for human-readable representations
- CQN is an object notation for machine-readable representations

And for Expressions:

- CXL is a syntax for human-readable representations
- CXN is an object notation for machine-readable representations

...

What is a CDS Model?

Models in *cds* are plain JavaScript objects conforming to the *Core Schema Notation (CSN)*. They can be parsed from *.cds* sources, read from *.json* or *.yaml* files or dynamically created in code at runtime.

The following ways and examples of creating models are equivalent:

In Plain Coding at Runtime

```js
const cds = require('@sap/cds')

// define the model
var model = {definitions:{
    Products: {kind:'entity', elements:{
        ID: {type:'Integer', key:true},
        title: {type:'String', length:11, localized:true},
        description: {type:'String', localized:true},
    }},
    Orders: {kind:'entity', elements:{
        product: {type:'Association', target:'Products'},
        quantity: {type:'Integer'},
    }},
}}

// do something with it
console.log (cds.compile.to.yaml (model))
```

Parsed at Runtime

```js
const cds = require('@sap/cds')

// define the model
var model = cds.parse (`
    entity Products {
        key ID: Integer;
        title: localized String(11);
        description: localized String;
    }
    entity Orders {
        product: Association to Products;
        quantity: Integer;
    }
`)

// do something with it
console.log (cds.compile.to.yaml (model))
```

From *.cds* Source Files

```cds
// some.cds source file
entity Products {
    key ID: Integer;
    title: localized String(11);
    description: localized String;
}
entity Orders {
    product: Association to Products;
    quantity: Integer;
}
```

Read/parse it, and do something with it, for example:

```js
const cds = require('@sap/cds')
cds.get('./some.cds') .then (cds.compile.to.yaml) .then (console.log)
```

> Which is equivalent to: *`cds ./some.cds -2 yaml`* using the CLI

From *.json* Files

```json
{"definitions": {
    "Products": {
        "kind": "entity",
        "elements": {
            "ID": { "type": "Integer", "key": true },
            "title": { "type": "String", "length": 11, "localized": true },
            "description": { "type": "String", "localized": true }
        }
    },
    "Orders": {
        "kind": "entity",
        "elements": {
            "product": { "type": "Association", "target": "Products" },
            "quantity": { "type": "Integer" }
        }
    }
}}
```

```js
const cds = require('@sap/cds')
cds.get('./some.json') .then (cds.compile.to.yaml) .then (console.log)
```

From Other Frontends

You can add any other frontend instead of using CDL; it's just about generating the respective CSN structures, most easily as *.json*. For example, different parties already added these frontends:

- ABAP CDS 2 csn
- OData EDMX 2 csn
- Fiori annotation.xml 2 csn
- i18n properties files 2 csn
- Java/JPA models 2 csn

Processing Models

All model processing and compilation steps, which can be applied subsequently just work on the basis of plain CSN objects. There's no assumption about and no lock-in to a specific source format.