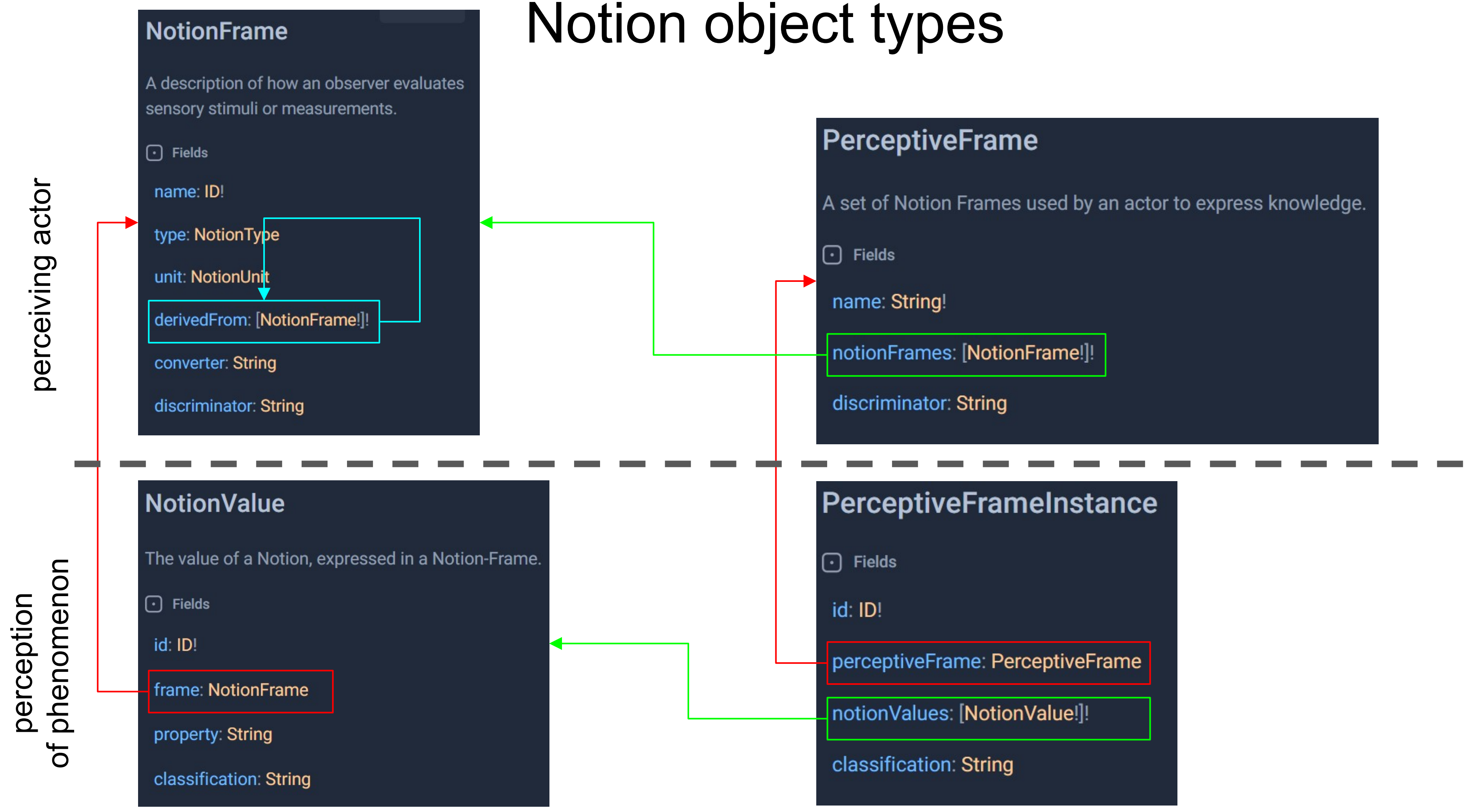
Notions API

Python/GraphQL implementation

2024-08-15

Notion object types



NotionFrame

A description of how an observer evaluates sensory stimuli or measurements.

Fields

name: ID!

type: NotionType

unit: NotionUnit

derivedFrom: [NotionFrame!]!

converter: String

discriminator: String

PerceptiveFrame

A set of Notion Frames used by an actor to express knowledge.

Fields

name: String!

notionFrames: [NotionFrame!]!

discriminator: String

NotionValue

The value of a Notion, expressed in a Notion-Frame.

Fields

id: ID!

frame: NotionFrame

property: String

classification: String

PerceptiveFrameInstance

Fields

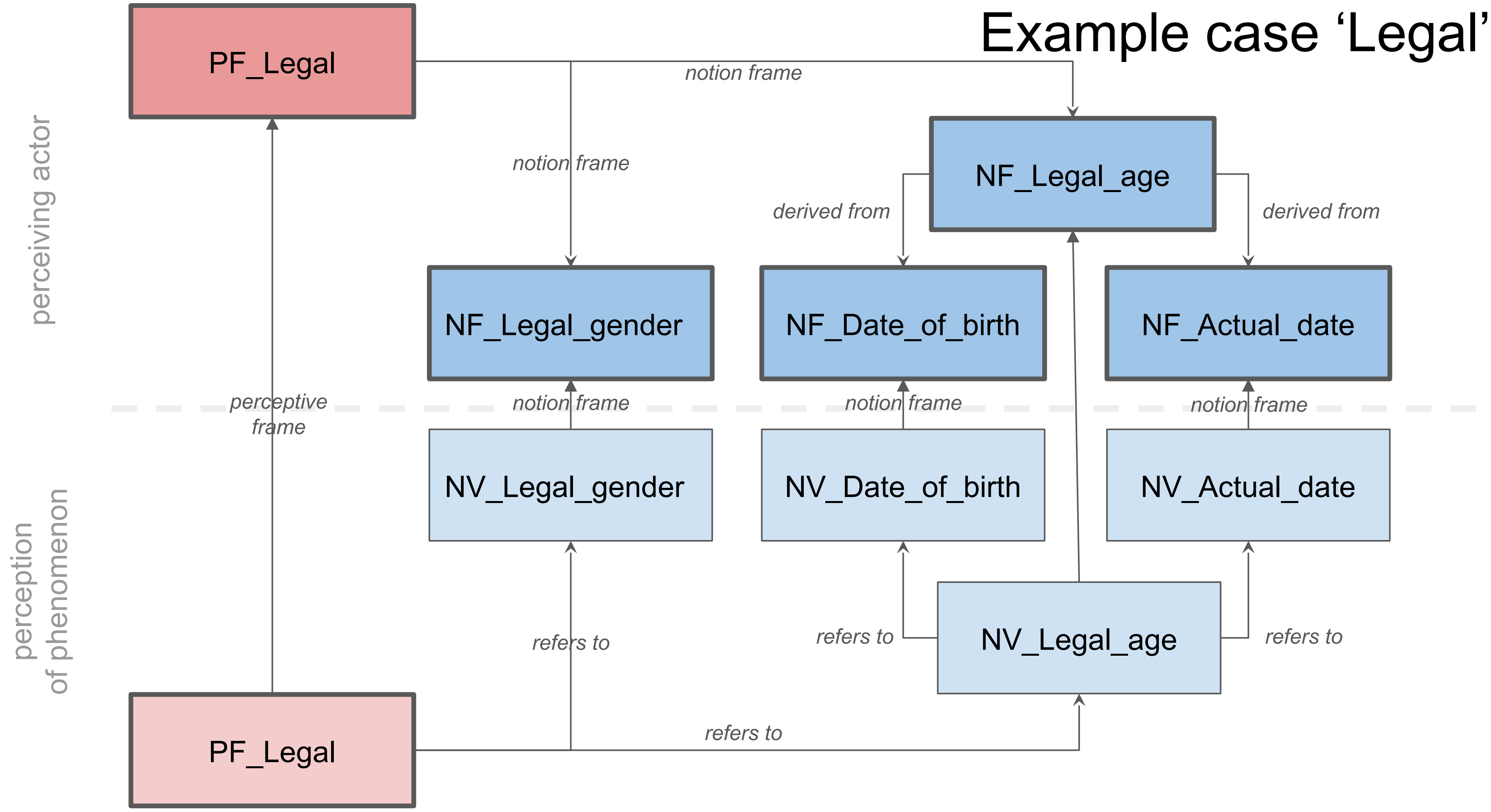
id: ID!

perceptiveFrame: PerceptiveFrame

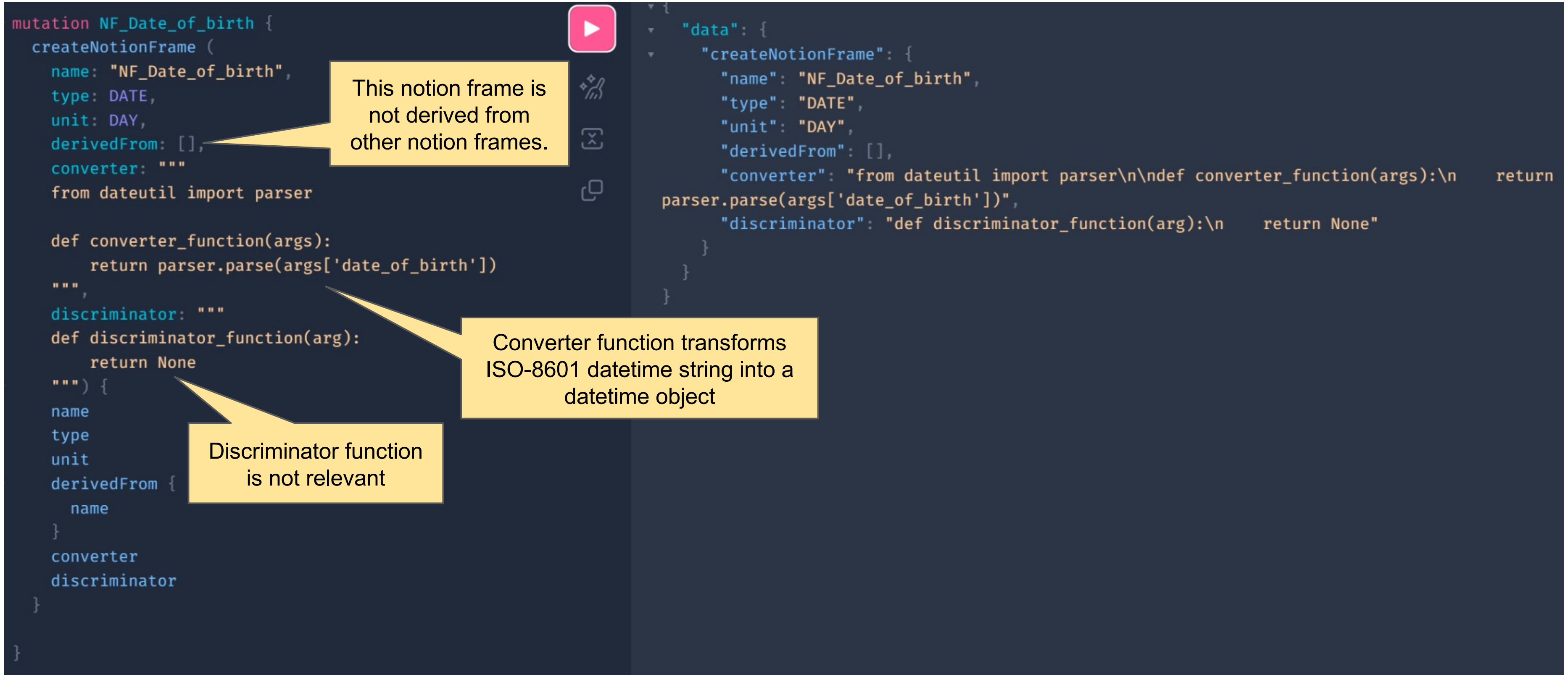
notionValues: [NotionValue!]!

classification: String

Example case 'Legal'



NF_Date_of_birth



```
mutation NF_Date_of_birth {
  createNotionFrame (
    name: "NF_Date_of_birth",
    type: DATE,
    unit: DAY,
    derivedFrom: [],
    converter: """
    from dateutil import parser

    def converter_function(args):
        return parser.parse(args['date_of_birth'])
    """,
    discriminator: """
    def discriminator_function(arg):
        return None
    """) {
    name
    type
    unit
    derivedFrom {
      name
    }
    converter
    discriminator
  }

}
```

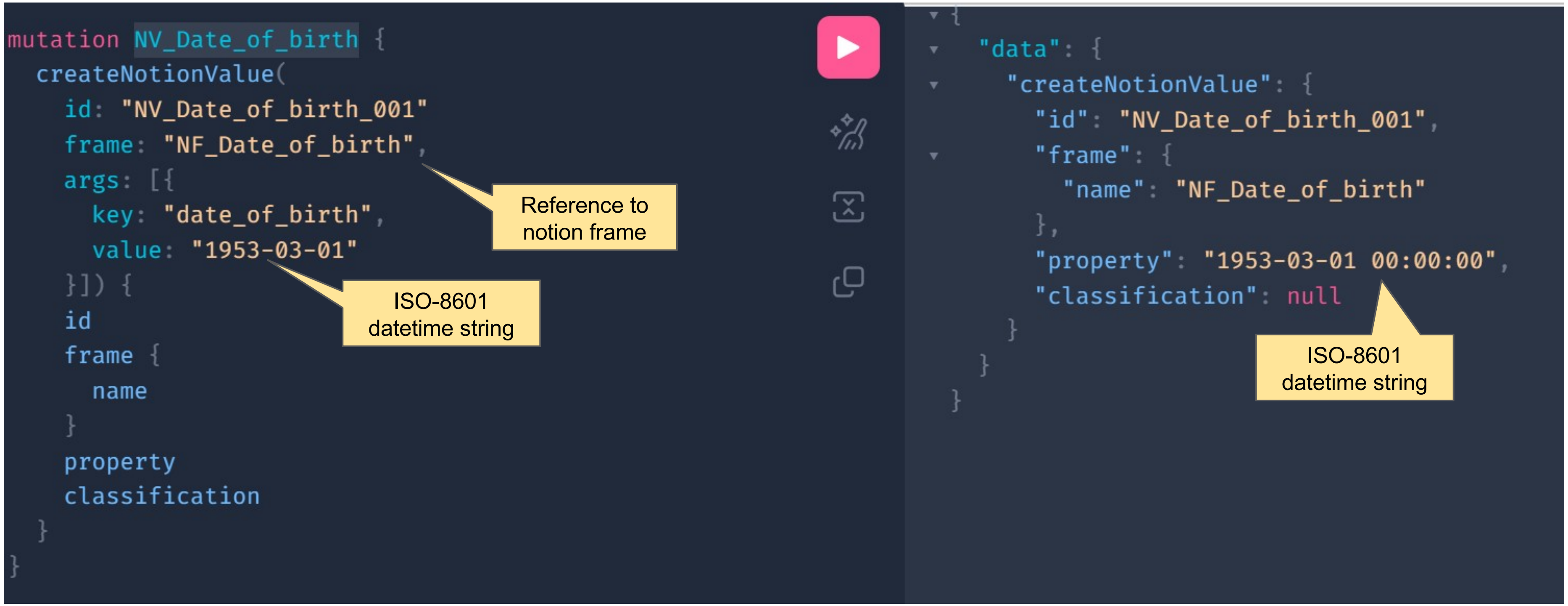
```
{
  "data": {
    "createNotionFrame": {
      "name": "NF_Date_of_birth",
      "type": "DATE",
      "unit": "DAY",
      "derivedFrom": [],
      "converter": "from dateutil import parser\n\ndef converter_function(args):\n    return parser.parse(args['date_of_birth'])",
      "discriminator": "def discriminator_function(arg):\n    return None"
    }
  }
}
```

This notion frame is not derived from other notion frames.

Converter function transforms ISO-8601 datetime string into a datetime object

Discriminator function is not relevant

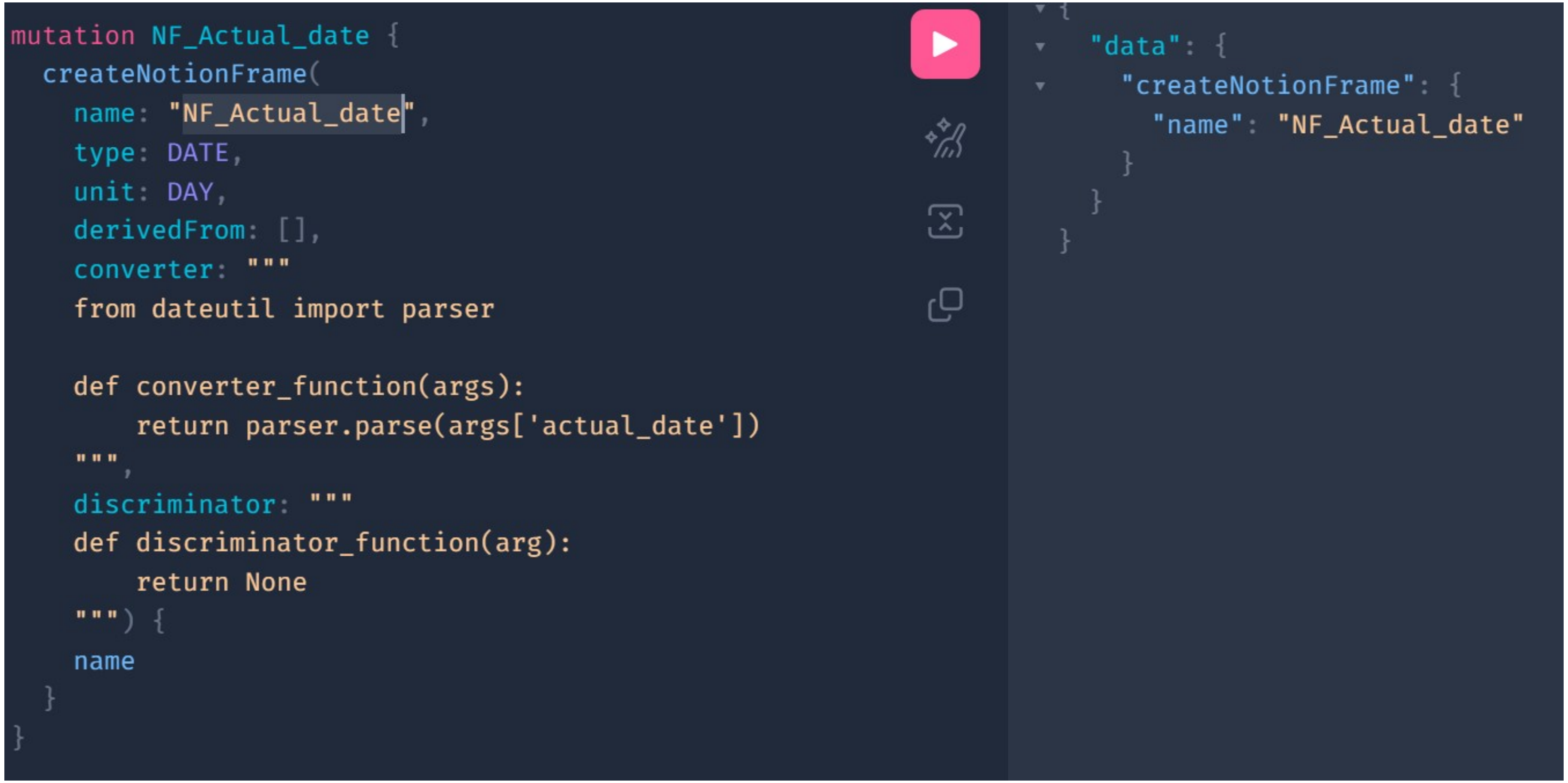
NV_Date_of_birth



```
mutation NV_Date_of_birth {
  createNotionValue(
    id: "NV_Date_of_birth_001"
    frame: "NF_Date_of_birth",
    args: [{
      key: "date_of_birth",
      value: "1953-03-01"
    }]) {
    id
    frame {
      name
    }
    property
    classification
  }
}
```

```
{
  "data": {
    "createNotionValue": {
      "id": "NV_Date_of_birth_001",
      "frame": {
        "name": "NF_Date_of_birth"
      },
      "property": "1953-03-01 00:00:00",
      "classification": null
    }
  }
}
```

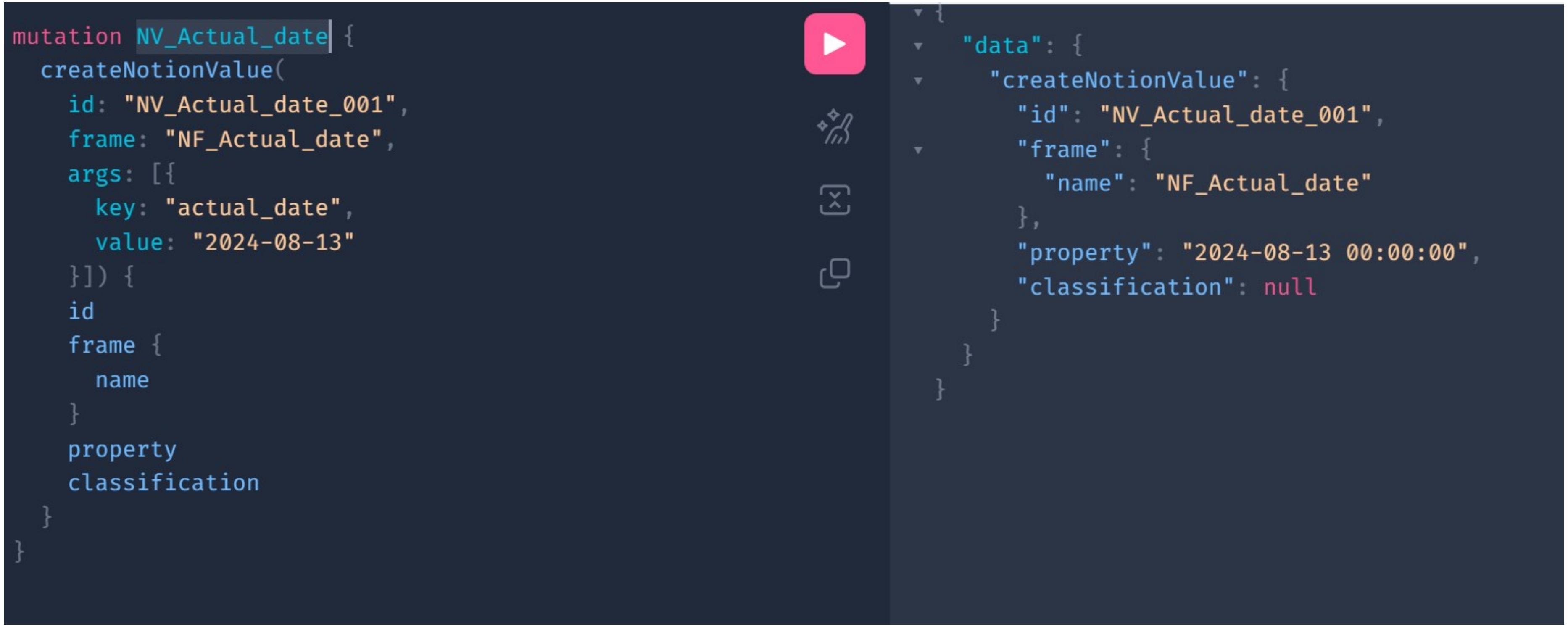
NF_Actual_date



```
mutation NF_Actual_date {
  createNotionFrame(
    name: "NF_Actual_date",
    type: DATE,
    unit: DAY,
    derivedFrom: [],
    converter: """
    from dateutil import parser

    def converter_function(args):
        return parser.parse(args['actual_date'])
    """,
    discriminator: """
    def discriminator_function(arg):
        return None
    """) {
    name
  }
}
```

```
{
  "data": {
    "createNotionFrame": {
      "name": "NF_Actual_date"
    }
  }
}
```

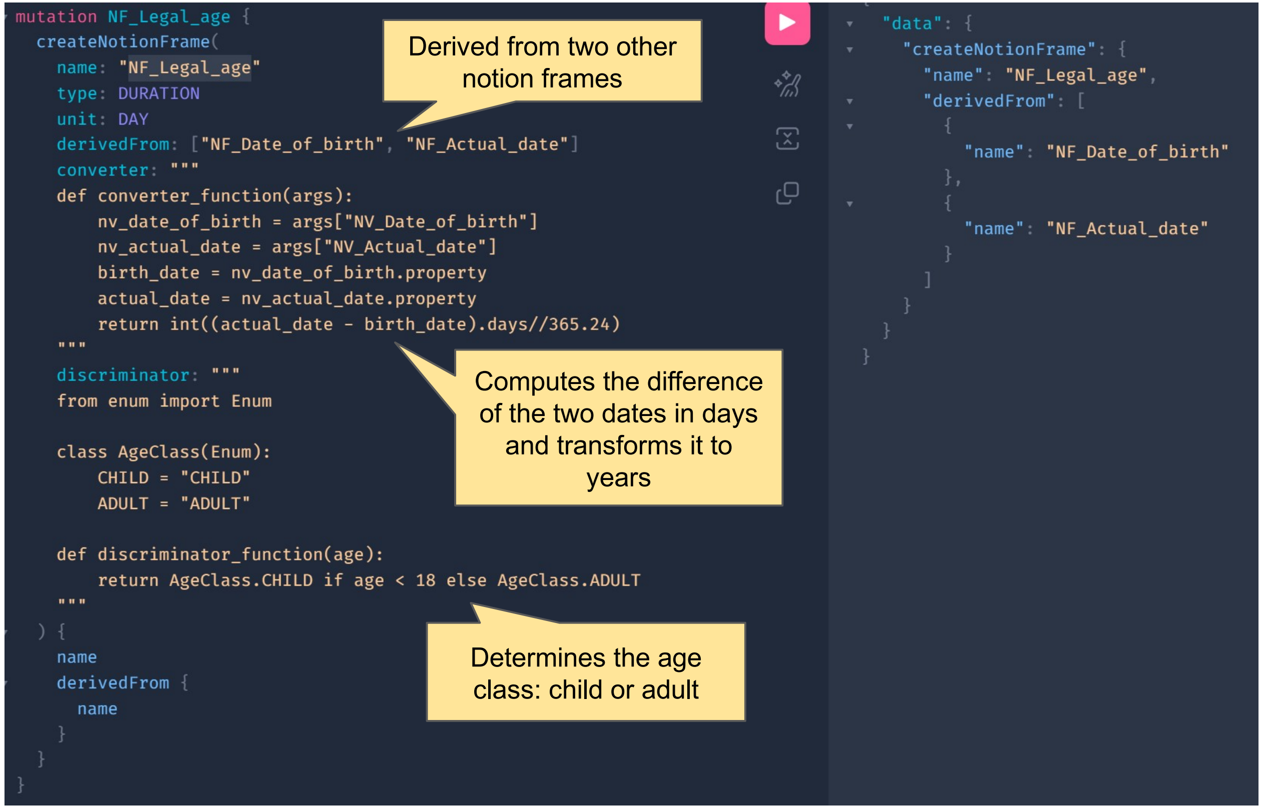
NV_Actual_date



```
mutation NV_Actual_date {
  createNotionValue(
    id: "NV_Actual_date_001",
    frame: "NF_Actual_date",
    args: [{
      key: "actual_date",
      value: "2024-08-13"
    }]) {
    id
    frame {
      name
    }
    property
    classification
  }
}
```

```
{
  "data": {
    "createNotionValue": {
      "id": "NV_Actual_date_001",
      "frame": {
        "name": "NF_Actual_date"
      },
      "property": "2024-08-13 00:00:00",
      "classification": null
    }
  }
}
```

NF_Legal_age

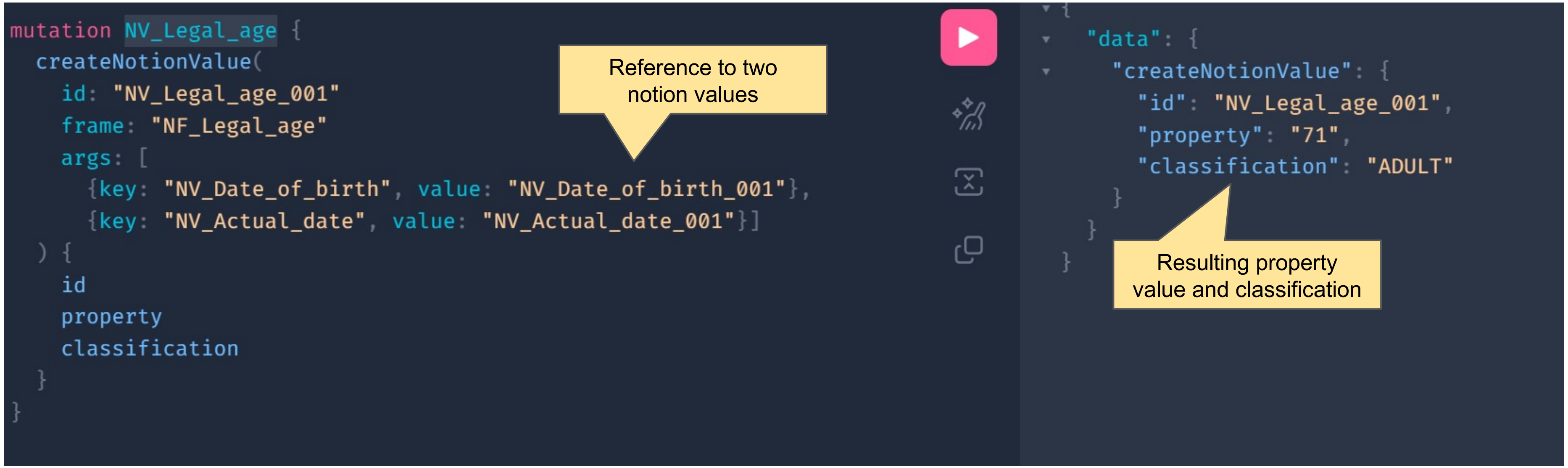


```
mutation NF_Legal_age {
  createNotionFrame(
    name: "NF_Legal_age"
    type: DURATION
    unit: DAY
    derivedFrom: ["NF_Date_of_birth", "NF_Actual_date"]
    converter: """
    def converter_function(args):
        nv_date_of_birth = args["NV_Date_of_birth"]
        nv_actual_date = args["NV_Actual_date"]
        birth_date = nv_date_of_birth.property
        actual_date = nv_actual_date.property
        return int((actual_date - birth_date).days//365.24)
    """
    discriminator: """
    from enum import Enum

    class AgeClass(Enum):
        CHILD = "CHILD"
        ADULT = "ADULT"

    def discriminator_function(age):
        return AgeClass.CHILD if age < 18 else AgeClass.ADULT
    """
  ) {
    name
    derivedFrom {
      name
    }
  }
}
```

```
{
  "data": {
    "createNotionFrame": {
      "name": "NF_Legal_age",
      "derivedFrom": [
        {
          "name": "NF_Date_of_birth"
        },
        {
          "name": "NF_Actual_date"
        }
      ]
    }
  }
}
```

Derived from two other notion frames
Computes the difference of the two dates in days and transforms it to years
Determines the age class: child or adult

NV_Legal_age



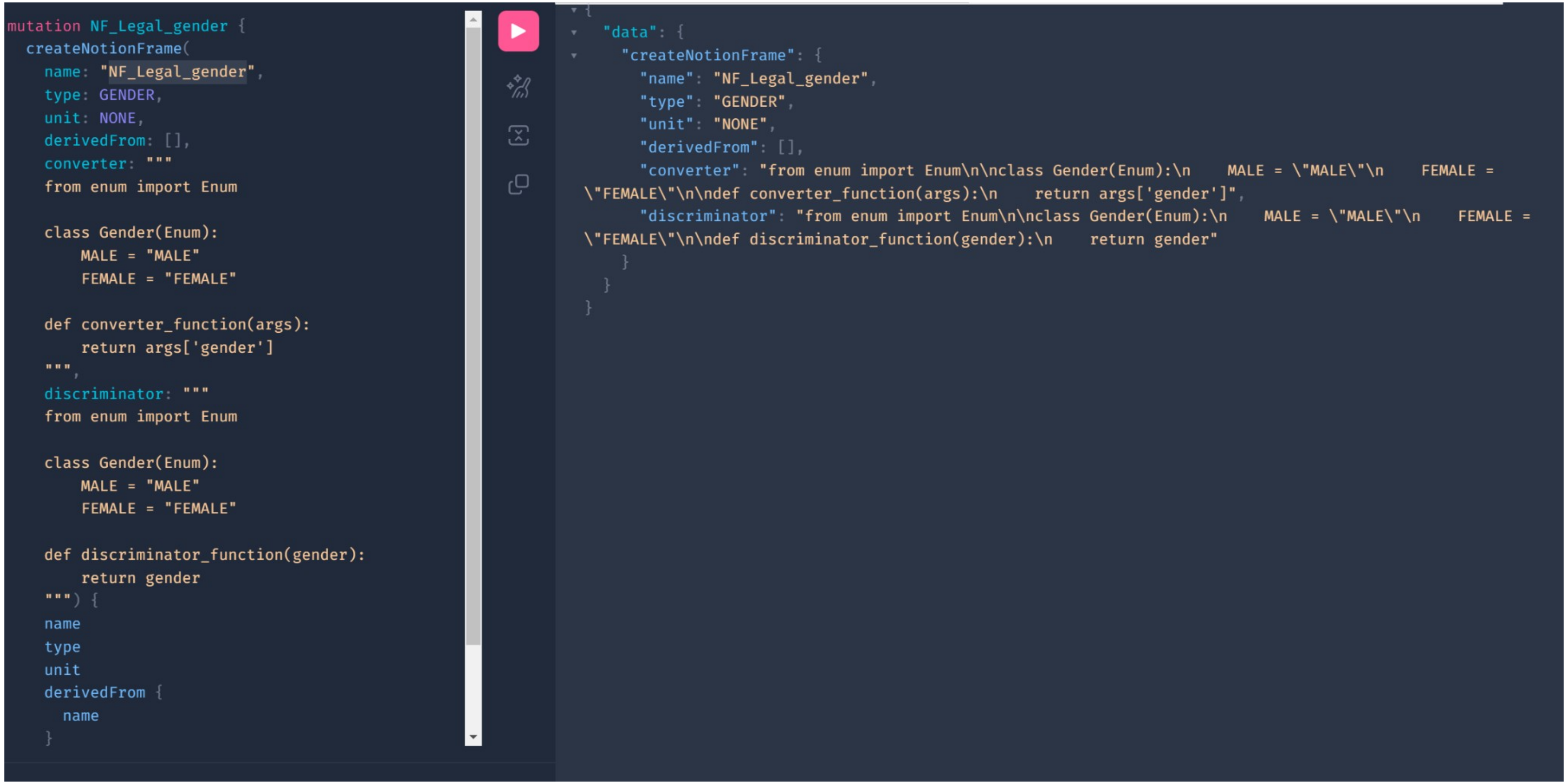
```
mutation NV_Legal_age {
  createNotionValue(
    id: "NV_Legal_age_001"
    frame: "NF_Legal_age"
    args: [
      {key: "NV_Date_of_birth", value: "NV_Date_of_birth_001"},
      {key: "NV_Actual_date", value: "NV_Actual_date_001"}]
  ) {
    id
    property
    classification
  }
}
```

Reference to two notion values

```
{
  "data": {
    "createNotionValue": {
      "id": "NV_Legal_age_001",
      "property": "71",
      "classification": "ADULT"
    }
  }
}
```

Resulting property value and classification

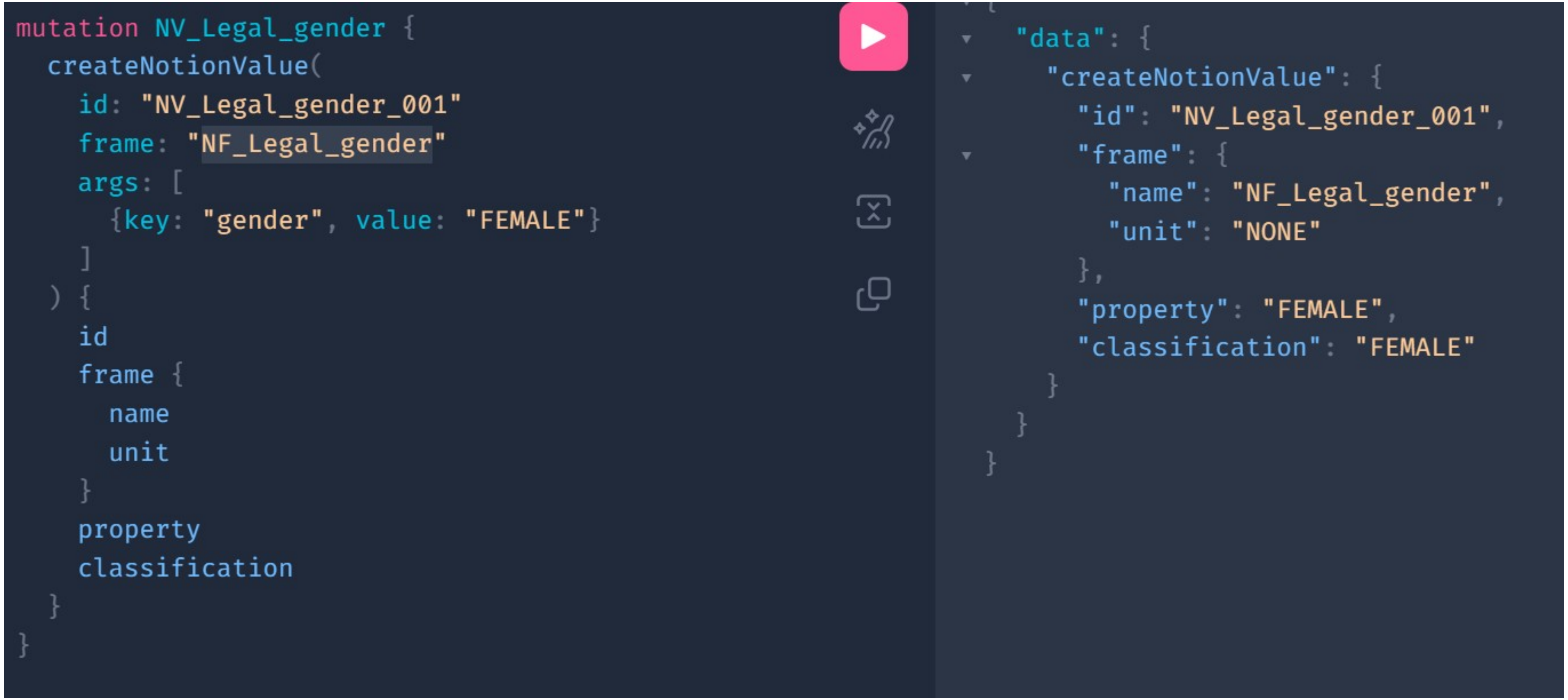
NF_Legal_gender



```
mutation NF_Legal_gender {
  createNotionFrame(
    name: "NF_Legal_gender",
    type: GENDER,
    unit: NONE,
    derivedFrom: [],
    converter: """
    from enum import Enum

    class Gender(Enum):
        MALE = "MALE"
        FEMALE = "FEMALE"

    def converter_function(args):
        return args['gender']
    """,
    discriminator: """
    from enum import Enum

    class Gender(Enum):
        MALE = "MALE"
        FEMALE = "FEMALE"

    def discriminator_function(gender):
        return gender
    """) {
    name
    type
    unit
    derivedFrom {
      name
    }
```

```
{
  "data": {
    "createNotionFrame": {
      "name": "NF_Legal_gender",
      "type": "GENDER",
      "unit": "NONE",
      "derivedFrom": [],
      "converter": "from enum import Enum\n\nclass Gender(Enum):\n    MALE = \"MALE\"\n    FEMALE = \"FEMALE\"\n\ndef converter_function(args):\n    return args['gender']",
      "discriminator": "from enum import Enum\n\nclass Gender(Enum):\n    MALE = \"MALE\"\n    FEMALE = \"FEMALE\"\n\ndef discriminator_function(gender):\n    return gender"
    }
  }
}
```

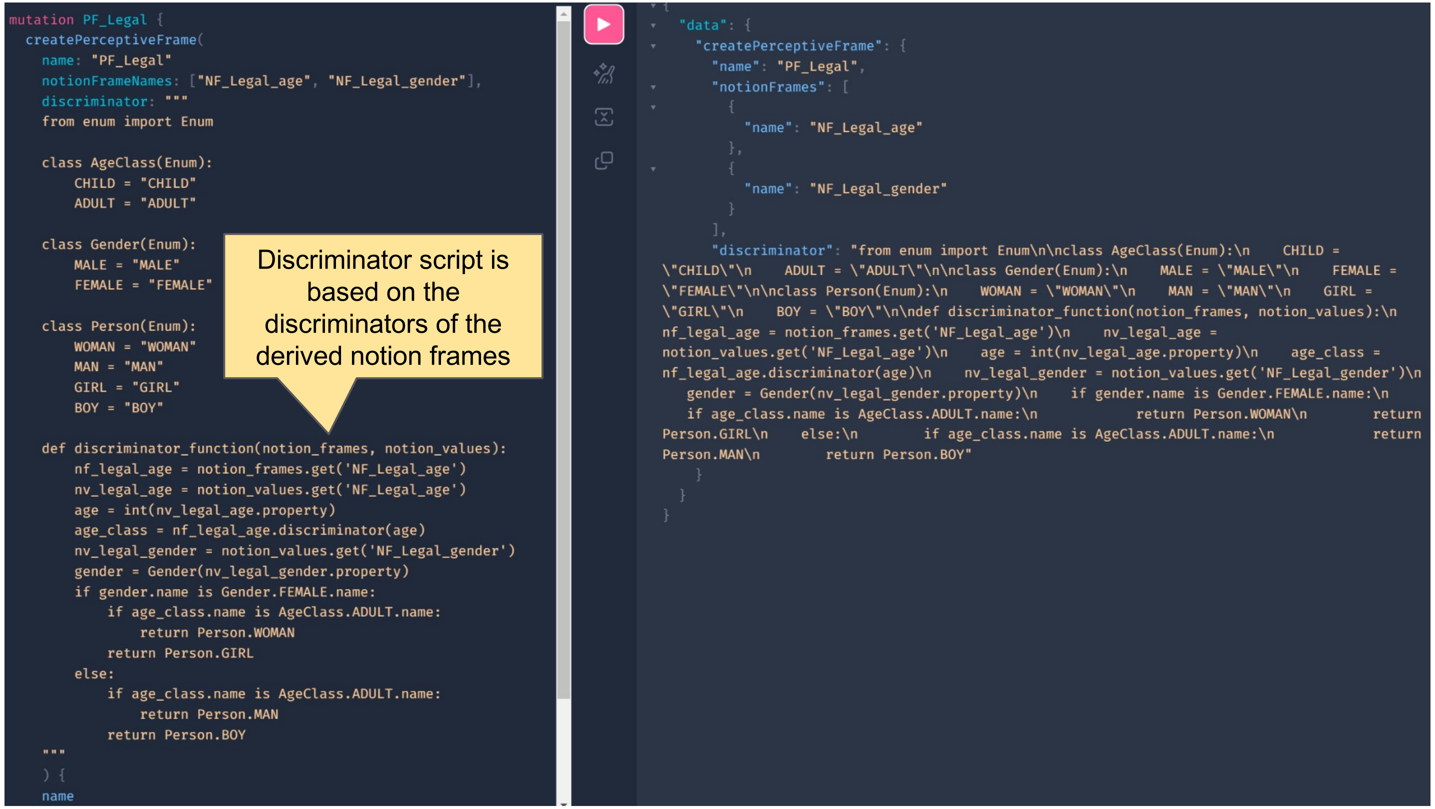
NV_Legal_gender



```
mutation NV_Legal_gender {
  createNotionValue(
    id: "NV_Legal_gender_001"
    frame: "NF_Legal_gender"
    args: [
      {key: "gender", value: "FEMALE"}
    ]
  ) {
    id
    frame {
      name
      unit
    }
    property
    classification
  }
}
```

```
  "data": {
    "createNotionValue": {
      "id": "NV_Legal_gender_001",
      "frame": {
        "name": "NF_Legal_gender",
        "unit": "NONE"
      },
      "property": "FEMALE",
      "classification": "FEMALE"
    }
  }
}
```

PF_Legal

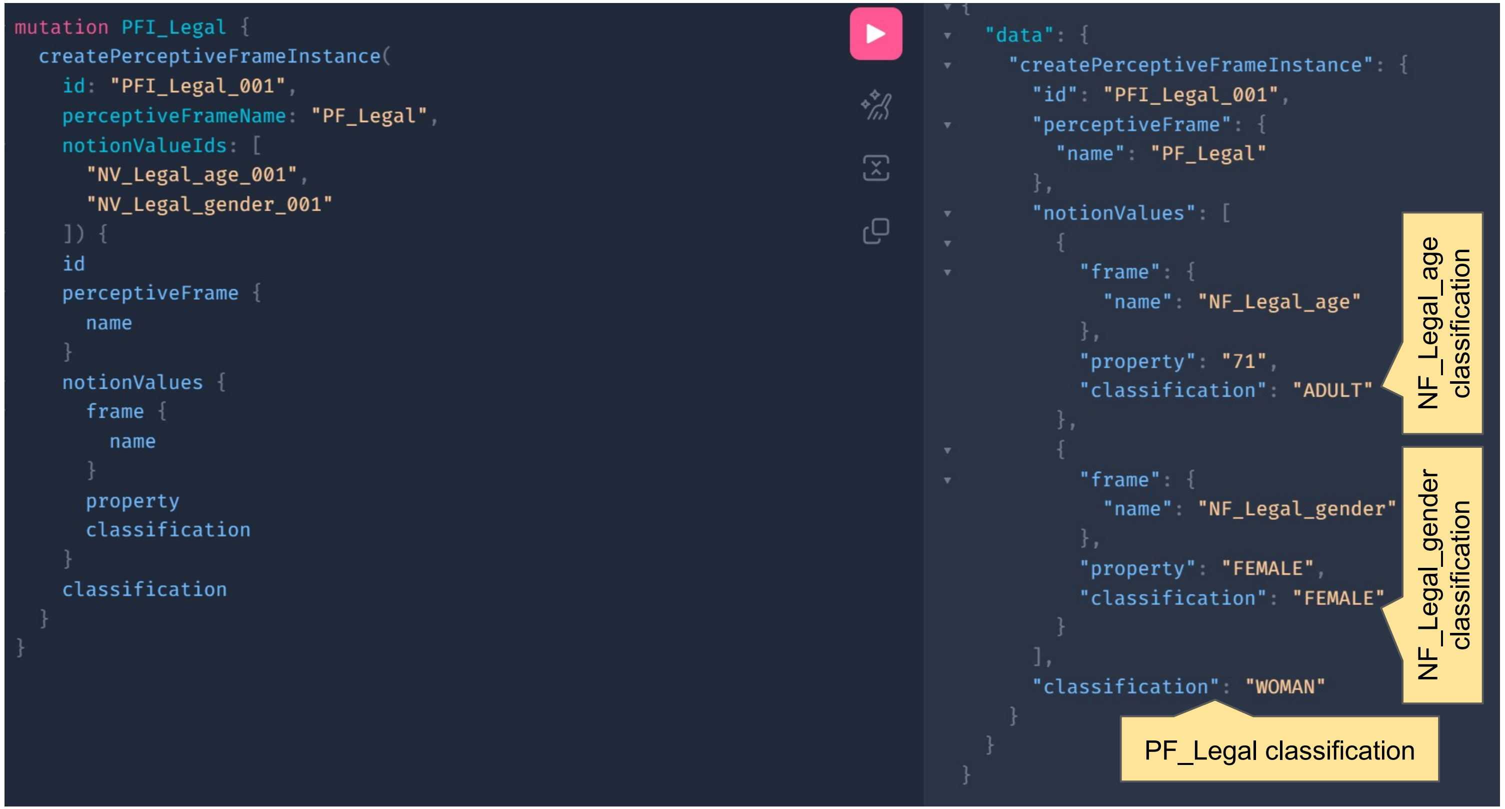


```
mutation PF_Legal {
  createPerceptiveFrame(
    name: "PF_Legal"
    notionFrameNames: ["NF_Legal_age", "NF_Legal_gender"],
    discriminator: """
    from enum import Enum

    class AgeClass(Enum):
        CHILD = "CHILD"
        ADULT = "ADULT"

    class Gender(Enum):
        MALE = "MALE"
        FEMALE = "FEMALE"

    class Person(Enum):
        WOMAN = "WOMAN"
        MAN = "MAN"
        GIRL = "GIRL"
        BOY = "BOY"

    def discriminator_function(notion_frames, notion_values):
        nf_legal_age = notion_frames.get('NF_Legal_age')
        nv_legal_age = notion_values.get('NF_Legal_age')
        age = int(nv_legal_age.property)
        age_class = nf_legal_age.discriminator(age)
        nv_legal_gender = notion_values.get('NF_Legal_gender')
        gender = Gender(nv_legal_gender.property)
        if gender.name is Gender.FEMALE.name:
            if age_class.name is AgeClass.ADULT.name:
                return Person.WOMAN
            return Person.GIRL
        else:
            if age_class.name is AgeClass.ADULT.name:
                return Person.MAN
            return Person.BOY
    """
    ) {
    name
```

Discriminator script is based on the discriminators of the derived notion frames

```
{
  "data": {
    "createPerceptiveFrame": {
      "name": "PF_Legal",
      "notionFrames": [
        {
          "name": "NF_Legal_age"
        },
        {
          "name": "NF_Legal_gender"
        }
      ],
      "discriminator": "from enum import Enum\n\nclass AgeClass(Enum):\n    CHILD = \"CHILD\"\n    ADULT = \"ADULT\"\n\nclass Gender(Enum):\n    MALE = \"MALE\"\n    FEMALE = \"FEMALE\"\n\nclass Person(Enum):\n    WOMAN = \"WOMAN\"\n    MAN = \"MAN\"\n    GIRL = \"GIRL\"\n    BOY = \"BOY\"\n\ndef discriminator_function(notion_frames, notion_values):\n    nf_legal_age = notion_frames.get('NF_Legal_age')\n    nv_legal_age = notion_values.get('NF_Legal_age')\n    age = int(nv_legal_age.property)\n    age_class = nf_legal_age.discriminator(age)\n    nv_legal_gender = notion_values.get('NF_Legal_gender')\n    gender = Gender(nv_legal_gender.property)\n    if gender.name is Gender.FEMALE.name:\n        if age_class.name is AgeClass.ADULT.name:\n            return Person.WOMAN\n        return Person.GIRL\n    else:\n        if age_class.name is AgeClass.ADULT.name:\n            return Person.MAN\n        return Person.BOY"
    }
  }
}
```

PFI_Legal



```
mutation PFI_Legal {
  createPerceptiveFrameInstance(
    id: "PFI_Legal_001",
    perceptiveFrameName: "PF_Legal",
    notionValueIds: [
      "NV_Legal_age_001",
      "NV_Legal_gender_001"
    ]) {
    id
    perceptiveFrame {
      name
    }
    notionValues {
      frame {
        name
      }
      property
      classification
    }
    classification
  }
}
```

```
{
  "data": {
    "createPerceptiveFrameInstance": {
      "id": "PFI_Legal_001",
      "perceptiveFrame": {
        "name": "PF_Legal"
      },
      "notionValues": [
        {
          "frame": {
            "name": "NF_Legal_age"
          },
          "property": "71",
          "classification": "ADULT"
        },
        {
          "frame": {
            "name": "NF_Legal_gender"
          },
          "property": "FEMALE",
          "classification": "FEMALE"
        }
      ],
      "classification": "WOMAN"
    }
  }
}
```