FEEL - Contexts

What I Will Learn

What Will I Learn?

At the end of this course you will be able to:

- Understand the basic context data type
- Operate with various context expressions:
    - **Access the context**
    - **Filter a list of context elements**
    - **Project a list of context elements**
- Use various general context functions:
    - get value()
    - get entries()
    - context put()
    - context merge()

Context

Data Type

In the next lessons, we will cover the basics of the **context** data type in 🔗 FEEL used in Camunda, followed by an exploration of key list expressions and functions All these topics will be cover along the course.

Basic Concepts

A context data type is a powerful feature that allows you to define and manipulate collections of **key-value pairs**:

- **key**: is a string. Refer to the 🔗 Variable Names Conventions for valid key names.
- **value**: can be **any FEEL data type**, including numbers, strings, booleans, lists, other contexts, and **FEEL expressions**.

They are similar to *JSON objects* or *dictionaries* in other programming languages, providing a structured way to handle complex data.

Examples

Creating a context data type

To create a context data type in FEEL, you enclose the elements within curly braces {}.

```
// empty context
{}

// context example
{
  "firstName": "John",
  "lastName": "Doe",
  "age": 25,
  "isEmployed": true,
```

```
  "address": {
    "street": "Main Street",
    "city": "Metropolis"
  },
  "scores": [80, 85, 90],
  "resultScores": sum(scores)
}
```

Creating a context using an expression

A context data type can also be created **through an expression** in a similar way.

Context:

```
{}
```

Expression:

```
{
  "person": {
    "name": "Alice",
    "birthday": "1981-03-04",
    "age": now().year - date(birthday).year
  }
}
```

Take a look at the "age" element, it's using an advanced temporal expression to calculate the age based on the birthday value.

Keys

When using an **expression** to create a context, **keys** can be either a **name** or a string. **Reminder**: **context input** accepts only a JSON object, so you can only use string values for the keys.

The **expression** will look like this with names as **keys**:

```
{
  person: {
    name: "Alice",
    birthday: "1981-03-04",
    age: now().year - date(birthday).year
  }
}
```

The **expression** will return the context data type itself:

Result:

```
{
  "person": {
    "name": "Alice",
    "birthday": "1981-03-04",
    "age": 43
  }
}
```

Expressions

In this lesson, we will explore various context expressions in FEEL. We will cover:

- Accessing the context
- Filtering a list of context elements
- Projecting a list of context elements

By practicing these expressions, you will become proficient in handling context manipulations in your decision models using FEEL.

Accessing the Context

Accessing the content of a provided content data

To access values in a provided context data, use the **dot notation** in your expression.

Context:

```
{
  "person": {
    "name": "Alice",
    "age": 30
  }
}
```

Expression:

person.name

Result:

"Alice"

Accessing the content of a declared content data

To access values of a context declared using an **expression**, use the **dot notation** after the context declaration.

Context:

```
{}
```

Expression:

```
{
  "person": {
    "name": "Alice",
    "age": 30
  }
}.person.name
```

Result:

```
"Alice"
```

Filtering a list of context elements

You can **filter** a list of context elements based on a **condition** using a filter expression.

Get the context elements where age is greater than 18

Expression:

```
[
  {
    name: "Jane",
    age: 22
  },
  {
    name: "William",
    age: 16
  }
][age > 18]
```

Result:

```
[
  {
    "name": "Jane",
    "age": 22
  }
]
```

Projecting a list of context elements

You can **project** a list of context elements based on the their **key**.

Project the name of the context elements

Expression:

```
[
  {
    name: "Jane",
    age: 22
  },
  {
    name: "William",
    age: 16
  }
].name
```

Result:

```
[
  "Jane",
  "William"
]
```

Context Functions

Get Value

The get value() function in 🔗 FEEL used in Camunda is used to retrieve a value from a context based either on a **single key** or a **list of keys** to provide the path to the value. This function is particularly useful when dealing with **simple and nested contexts** or when you need to dynamically access values.

Basic Concepts

// Syntax

get value(list, key)

get value(list, keys list)

The expression returns null for any attempt to retrieve a value using **non-existent keys**.

Retrieve a simple context key

Expression:

get value({color: "green"}, "color")

Result:

"green"

Retrieve a nested context key

Expression:

```
{
  "company": {
    "name": "Camunbankia",
    "offices": {
      "headquarters": {
```

```
      "city": "Neverland",
      "address": {
        "street": "5th Avenue",
        "number": 123
      }
    }
  }
},
"streetName": get value(company, ["offices", "headquarters", "address", "street"])
}.streetName
```

Result:

"5th Avenue"

Get Entries

The get entries() function in 🔗 FEEL used in Camunda is used to retrieve **all key-value pairs** from a context as a list of entries. Each entry is represented as a context with keys "**key**" and "**value**".

Basic Concepts

```
// Syntax
get entries(context)
```

Expression:

```
{
  "employee": {
    "id": 1001,
    "name": "Alice",
    "position": "Engineer"
  },
  "entries": get entries(employee)
}.entries
```

Result:

```
[
  {
    "key": "id",
    "value": 1001
  },
  {
    "key": "name",
    "value": "Alice"
```

```
  },
  {
    "key": "position",
    "value": "Engineer"
  }
]
```

Context Put

The context put() function in 🔗 FEEL used in Camunda allows you to **add** or **update** a **key-value pair** in a context. It returns a new context that includes the entry. This function is particularly useful for dynamically modifying contexts during decision-making processes.

A list of **2 keys** will define the path of the entry to add or update **nested contexts**:

- If keys contains the keys [k1, k2] then it adds the nested entry k1.k2 = value to the context
- If an entry for the same keys already exists in the context, it overrides the value.

Basic Concepts

// Syntax

context put(context, key, value)

context put(context, 2 keys, value)

Add a simple entry

Expression:

```
{
  "person": {
    "name": "Peter",
    "age": 30
  },
  "updatedPerson": context put(person, "city", "New York")
}.updatedPerson
```

Result:

```
{
  "name": "Peter",
  "age": 30,
  "city": "New York"
```

```
}
```

Modify a simple entry

Expression:

```
{
  "person": {
    "name": "Peter",
    "age": 30,
    "city": "New York"
  },
  "updatedPerson": context put(person, "age", 31)
}.updatedPerson
```

Result:

```
{
  "name": "Peter",
  "age": 31,
  "city": "New York"
}
```

Add a 2 keys entry

Context:

```
{
    "book": {
        "title": "1984",
        "author": "George Orwell"
    }
}
```

Expression:

```
context put(book, ["publication","year"], 1949)
```

Result:


```
{
  "author": "George Orwell",
  "title": "1984",
  "publication": {
    "year": 1949
  }
}
```

Context Merge

The context merge() function in 🔗 FEEL used in Camunda is useful for **consolidating data** from a list of contexts or for **enriching an existing context** with additional information.

Basic Concepts

// Syntax

context merge(context list)

Expression:

```
{
  "context1": {
    "name": "Alice",
    "age": 30
  },
  "context2": {
    "city": "New York",
    "age": 31
  },
  "mergedContext": context merge(context1, context2)
}.mergedContext
```

Result:

```
{
  "name": "Alice",
  "age": 31,
  "city": "New York"
}
```

Challenge

Instructions

Objective

In this challenge, you will use your knowledge of FEEL expressions to analyze a context containing data about teachers.

Here is the **context** you will be working with:


```
{
  "teachers": [
    {
      "name": "John Doe",
      "yearsOfExperience": 5,
      "skills": {
        "Classroom Management": "Intermediate",
        "Curriculum Development": "Basic",
        "Mathematical Analysis": "Expert"
      }
    },
    {
      "name": "Jane Smith",
      "yearsOfExperience": 7,
      "skills": {
        "Literary Criticism": "Expert",
        "Creative Writing": "Intermediate",
        "Public Speaking": "Basic"
      }
    },
```

{
  "name": "Emily Johnson",
  "yearsOfExperience": 4,
  "skills": {
    "Lab Safety": "Intermediate",
    "Biology Research": "Basic",
    "Student Engagement": "Intermediate"
  }
},
{
  "name": "Michael Brown",
  "yearsOfExperience": 10,
  "skills": {
    "Historical Research": "Expert",
    "Archival Studies": "Expert",
    "Lesson Planning": "Intermediate"
  }
},
{
  "name": "Jessica Garcia",
  "yearsOfExperience": 3,
  "skills": {
    "Chemical Analysis": "Basic",
    "Safety Procedures": "Intermediate",
    "Problem Solving": "Basic"
  }
},
{

"name": "William Martinez",
"yearsOfExperience": 6,
"skills": {
"Physical Fitness": "Expert",
"Sports Coaching": "Intermediate",
"Team Building": "Intermediate"
}
},
{
"name": "Sarah Wilson",
"yearsOfExperience": 9,
"skills": {
"Artistic Expression": "Expert",
"Design Fundamentals": "Intermediate",
"Art History": "Basic"
}
},
{
"name": "Brian Anderson",
"yearsOfExperience": 8,
"skills": {
"Theoretical Physics": "Expert",
"Experimental Design": "Intermediate",
"Quantitative Analysis": "Expert"
}
},
{
"name": "Nancy Davis",

```
    "yearsOfExperience": 2,
    "skills": {
      "Music Theory": "Intermediate",
      "Instrumental Skills": "Basic",
      "Performance Techniques": "Basic"
    }
  },
  {
    "name": "David Miller",
    "yearsOfExperience": 11,
    "skills": {
      "Programming": "Expert",
      "Algorithm Design": "Expert",
      "Cybersecurity": "Intermediate"
    }
  }
 ]
}
```


Task

Get the context **skills** from the **senior teachers** (with at least 10 years of experience)

Hints

Start by filtering the teachers with at least 10 years of experience

---

Get the skills of the senior teachers

Finally merge the skills contexts

Solution

Expression:

context merge(teachers[yearsOfExperience >= 10].skills)

Result:

{"Cybersecurity":"Intermediate","Historical Research":"Expert","Lesson Planning":"Intermediate","Algorithm Design":"Expert","Programming":"Expert","Archival Studies":"Expert"}

Review

Through this course, you have learned how to evaluate **FEEL Context** data type, expressions and functions used in Camunda.

What Did I Learn?

You should now be able to:

- Understand the basic context data type
- Operate with various context expressions:
    - **Access the context**
    - **Filter a list of context elements**
    - **Project a list of context elements**
- Use various general context functions:
    - get value()
    - get entries()
    - context put()
    - context merge()