IIP-Ecosphere

Next Level Ecosphere for
Intelligent Industrial Production

Service Integration: How to Configure Datatypes

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

IIP-Ecosphere Platform

Configure Datatypes

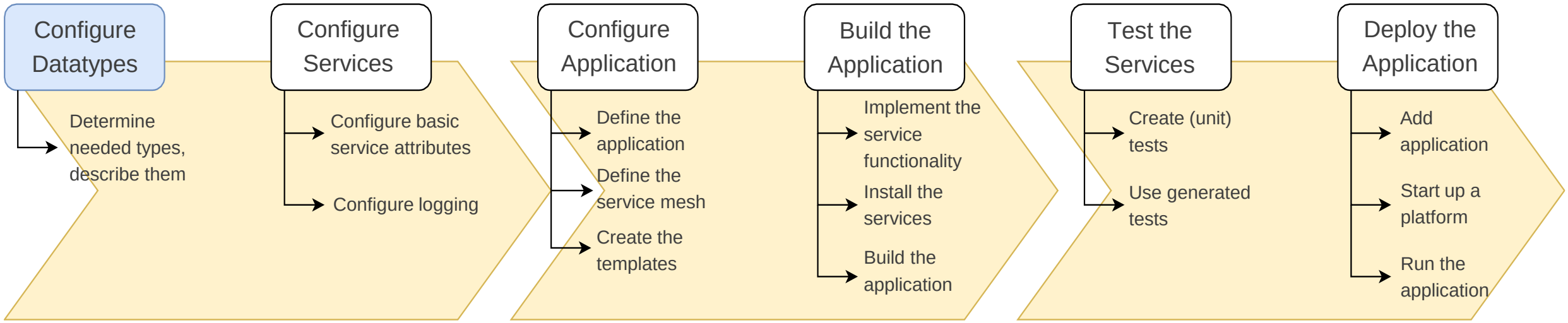


Table of Contents

Prerequisites

- Required:
  - Installed the platform and its dependencies or development container
  - Installed the IDE for IIP-Ecosphere Platform (provided Eclipse Version)
- Optional:
  - Introduction to code generation

Table of Contents

Introduction .ivml Structure

- The *.ivml* Files to configure the services and the application can be found in "*src/test/easy/….ivml*"
- ONLY the .ivml Files containing "*Part*" in their name and the "*TechnicalSetup*" need to be edited to create an application
- They each contain a "*Part*" of the complete application setup
- "*AllServices*", "*AllTypes*" import their "*Part*" files
- "*PlatformConfiguration*" combines the parts into one for generation

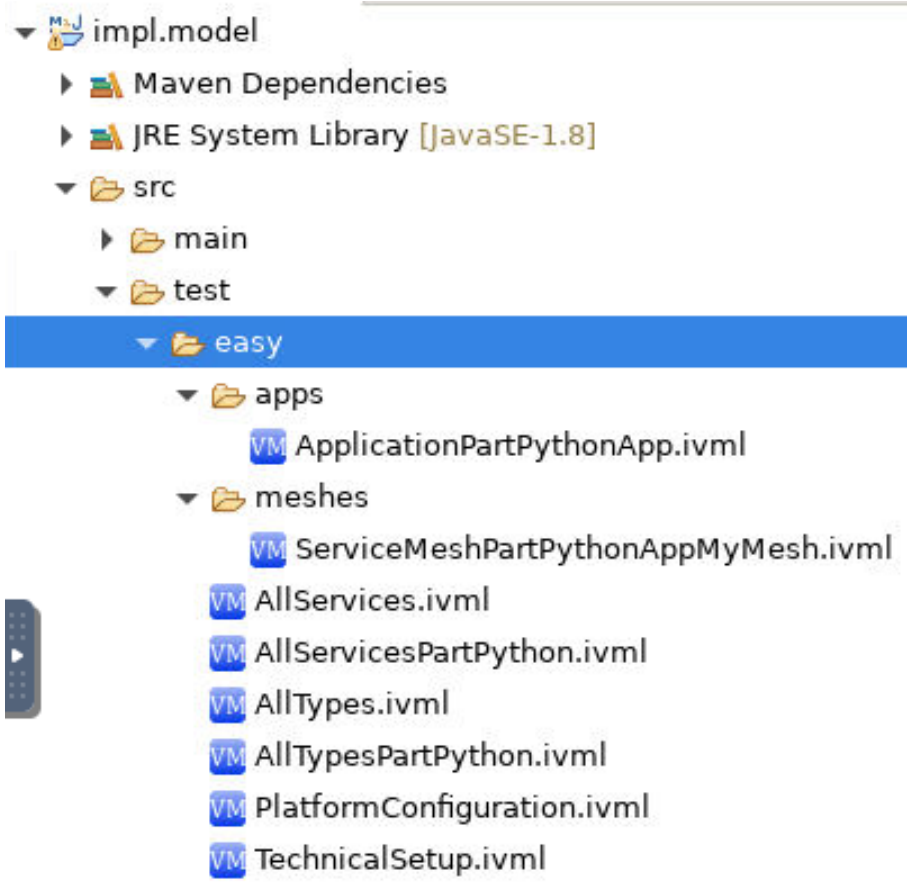
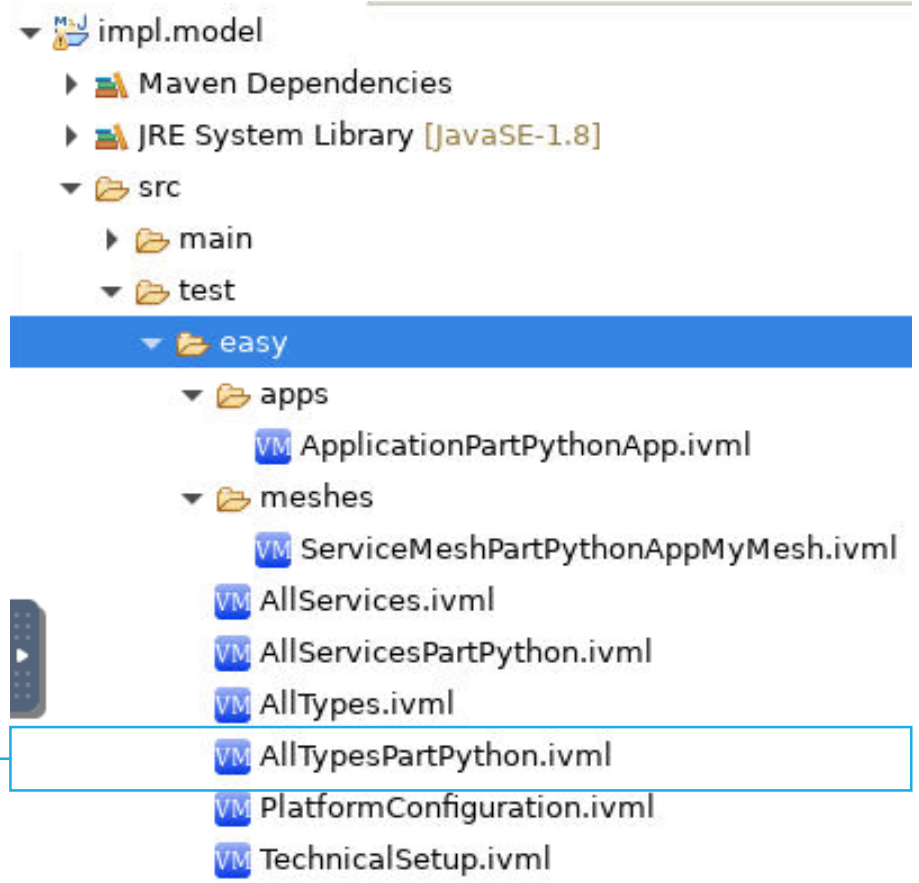


Table of Contents

Configuring Datatypes (1)

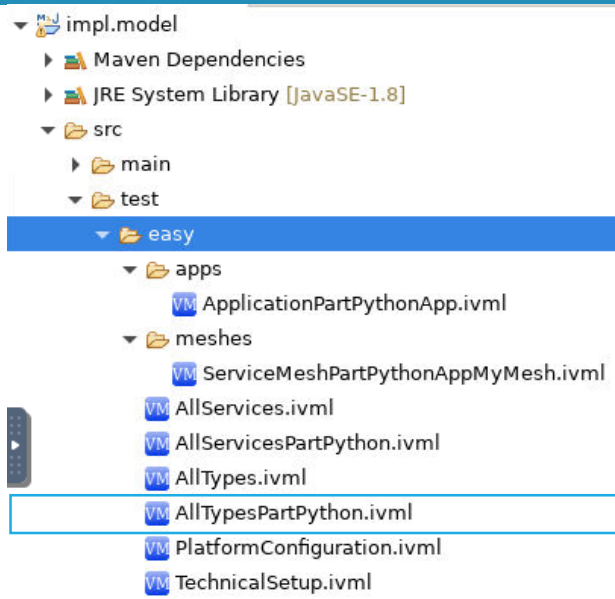
- The dev container will contain an example set of files
  - They are only supposed to be a guidance
  - Ultimately superseded by the UI



- Define the needed datatypes

Configuring Datatypes (2)

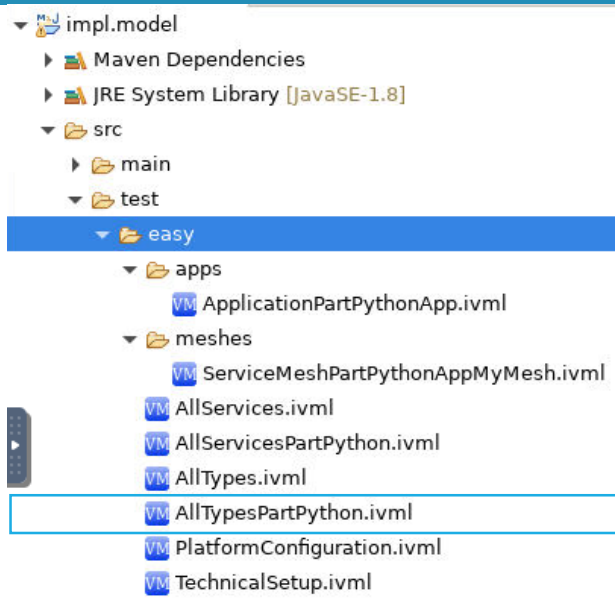
- Define the data types in the *AllTypesPart….ivml* file:
  - To transport data we package the different values into one object with corresponding members variables e.g. "*intExample*"




```
RecordType InData = {
    name = "InData",
    fields = {
        Field {
            name = "intExample",
            type = refBy(IntegerType)
        }, Field {
            name = "floatExample",
            type = refBy(FloatType)
        }, Field {
            name = "stringExample",
            type = refBy(StringType)
        }, Field {
            name = "doubleExample",
            type = refBy(DoubleType)
        }
    }
};
```

- Example data type containing an *int*, an *double*, an *float* and a *String*
- Corresponding classes will be translated to Java and Python (If needed)

Configuring Datatypes (3)



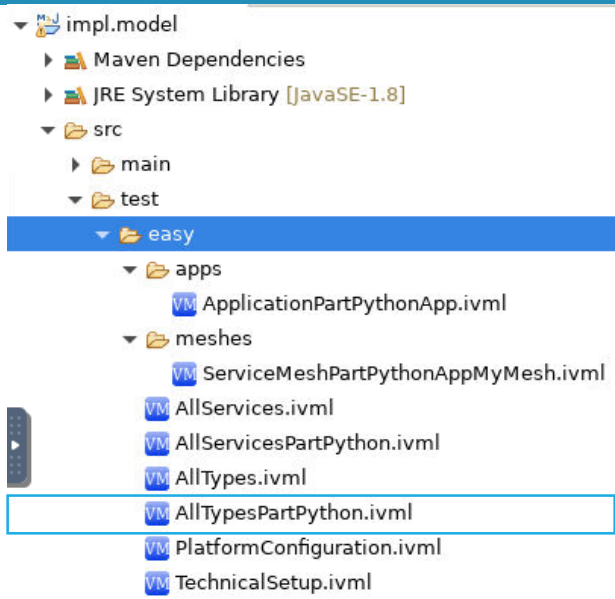
- We add a "*Field*" for each data point we want to send
- Always in the structure of *name = "<someName>"* and *type = refBy(<concreteDataType>)*
  - A list of all types can be found further on

```
RecordType InData = {
    name = "InData",
    fields = {
        Field {
            name = "intExample",
            type = refBy(IntegerType)
        }, Field {
            name = "floatExample",
            type = refBy(FloatType)
        }, Field {
            name = "stringExample",
            type = refBy(StringType)
        }, Field {
            name = "doubleExample",
            type = refBy(DoubleType)
        }
    }
};
```

- <> are used as placeholder for an actual value
- e.g. *<someName>* is a placeholder for the actual name of the datapoint to be send, *<concreteDataType>* is a placeholder for the one of the possible datatypes

Configuring Datatypes (4)

- Example
  - We input three numbers and string which could be three measured values and a product ID
  - We output a single number and the string




```
RecordType InData = {
    name = "InData",
    fields = {
        Field {
            name = "intExample",
            type = refBy(IntegerType)
        }, Field {
            name = "floatExample",
            type = refBy(FloatType)
        }, Field {
            name = "stringExample",
            type = refBy(StringType)
        }, Field {
            name = "doubleExample",
            type = refBy(DoubleType)
        }
    }
};
```

```
RecordType OutData = {
    name = "OutData",
    fields = {
        Field {
            name = "stringExample",
            type = refBy(StringType)
        }, Field {
            name = "result",
            type = refBy(DoubleType)
        }
    }
};
```

Configuring Datatypes (5)

- Reference of supported datatypes:
  - **Numerical**: IntegerType, ShortType, LongType, FloatType, DoubleType, ByteType
  - **String**: StringType, StringBase64Type, ByteStringType
  - **Arrays**: IntegerArrayType, ByteArrayType, DoubleArrayType
  - **OPC UA**: UnsignedInteger16Type, UnsignedInteger32Type, UnsignedInteger64Type, SByteType, Integer16Type, Integer32Typ, Integer64Type
  - ObjectType
  - DateTimeType
  - BooleanType
  - EnumType

Summary

- What we learned
  - How to create custom datatypes to transport values between services
  - What datatypes are available to us by default
- How to go on
  - How to edit the services
  - How to build an application
  - How to test an application