
P
C
P
G
P
C
P
G
P
G
S
G
S

# Table of Contents

# ESDL Documentation

## Current state

ESDL is still being heavily developed, with new versions being released periodically. We are using ESDL in many different projects now to get experience in using it and collect feedback. Some parts are still subject to discussion. Feel free to start using it, but some things will definitely change in the coming months.

# ESDL Documentation

This documentation contains the following chapters:

- Introduction: General introduction into what ESDL is and for what purposes it can be used.
- Energy Data Modelling: Explanation of the general concepts of Energy Data Modelling
- ESDL concepts: Explanation of the what and why of different parts of the ESDL language
- How to use ESDL: Explanation of how to use ESDL, practical tips
- Tooling for ESDL: Explanation of tooling to contribute to the developments of ESDL, to use ESDL to model an energy system and to integrate ESDL in your own tooling
- Examples: Several examples of how ESDL can be applied

Alternatively, you can download an eBook version as pdf, epub, mobi.

# Introduction

The energy transition from fossil to renewable requires a major change in the construction and the operation of the energy system. The energy system is a complex system with numerous relations and dependencies. Therefore, the impact of possible changes is hard to determine. For instance, the transition from heating with natural gas to electrical heating has impact on: gas transport network, electricity supply, required insolation level of build environment and many other facets of the energy system. In order to constructively reason on these changes of the energy system, an objective and complete information basis is necessary. This leads to an integral understanding of energy system. This document provides a description of the Energy System Description Language (ESDL), a first step to an integral understanding of the energy system.

# What is ESDL?

The Energy System Description Language (ESDL) is a modelling language created for modelling the components in an energy system and their relations towards each other. Furthermore ESDL is capable of expressing the dynamic behaviour of components in the energy system. For instance the power consumption of an neighbourhood. ESDL describes components by their basic functionality (so called *Energy Capabilities*), these are modelled in 5 abstract categories: Production, Consumption, Storage, Transport and Conversion. ESDL enables energy modelers to model a complex energy system in a generic way. The language is machine readable so makers of energy transition calculation tools and GIS applications can support ESDL in order to enforce the interoperability of their products.

# Application Areas of ESDL

ESDL can be used:

- by energy transition calculation tools: common language for energy transition calculation tools. To describe inputs and outputs of those tools.

This allows for integration of multiple tools.

- in an Energy Information System: ESDL can be used as a basis for a central energy information system where the energy system of a certain region is registered.
- as a language for (local) governments to model and share their (local) energy system.
- to monitor the evolution of an energy system: multiple ESDL snapshots of a certain area over time provide insight in the evolution of an energy system.
- as a format to share data relevant to energy systems or the energy transition. Examples:
  - CO2 emissions per energy carrier
  - Technology factsheets for specific components, brands, types (e.g. a heat pump factsheet that describes its typical parameters)
  - Cost information of assets, or expected cost developments in future
  - Standard configurations or templates of typical parts of the energy system (e.g. a house with a heat pump, solar panels and an EV charging station)

# Energy Data Modelling

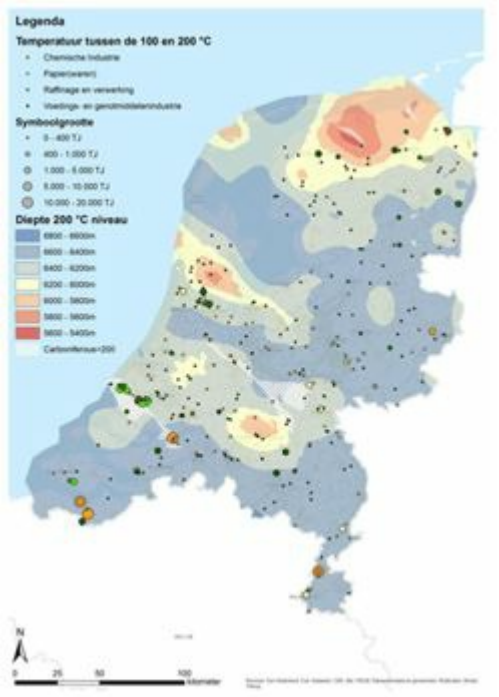
In order to fully understand how an energy system operates and how it can evolve, multiple types of information needs to be combined. Information on how and where the energy system is installed, what the installed base is, what the evolution potential is and how the energy system is used all form a part of the puzzle. This chapter describes these different types of data, a description on how they can be combined and modelled in ESDL follows in the next chapters.

# Energy System Registration

The first type of information is named the ‘energy system registration’, this is geo-oriented data that contains information such as: buildings characteristics, installed base of energy consuming/producing/converting appliances and topologies and other characteristics of energy transportation networks.

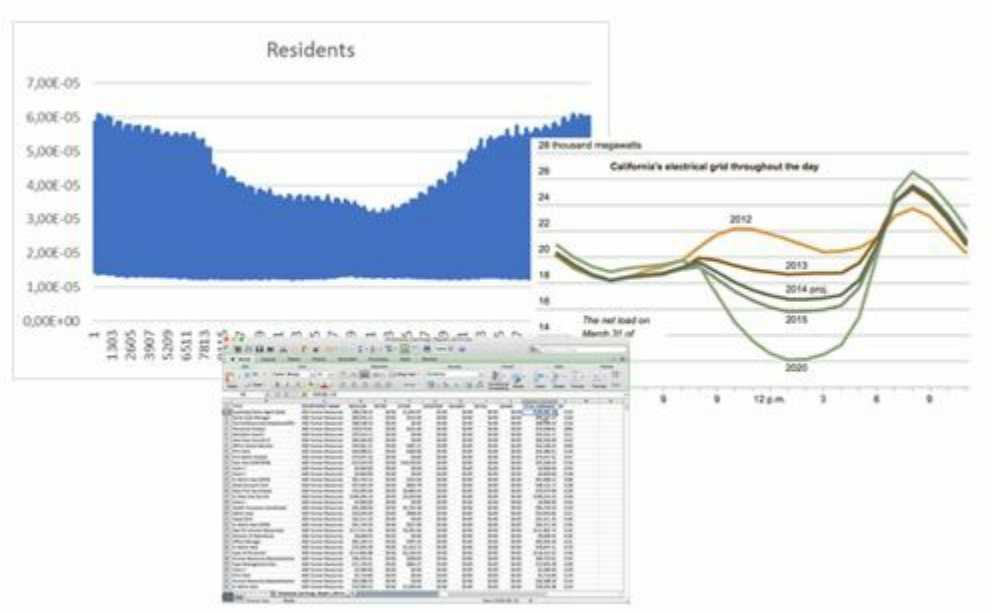
# Energy System Potential

The second type of information is the ‘energy system potential’. This information is also typically geo-oriented and describes the potential and constraints of the energy system to evolve. For example information about: areas where wind turbines can be installed, areas where geothermal sources can be found and areas that are suitable for district heating (heat networks).




# Energy System Usage

The third type of information is ‘energy system usage’. This type of information is providing insight on how the energy system is used over time. Typically this results in profiles or measurements for a certain duration of time. Examples of this information are: yearly energy consumption of a building, a profile of the production of a PV panel and a profile of the usage of a heat pump. This information describes the dynamic behaviour of the energy system.




# Combining data

To fully understand how a certain energy system is functioning and can evolve, the 3 types of information states in the sections above should be combined. Typically a source of energy data provides information within the scope of one of the types. ESDL provides the means to model these three aspects of the energy system in one common language and reason more powerful on the combined data.

# ESDL concepts

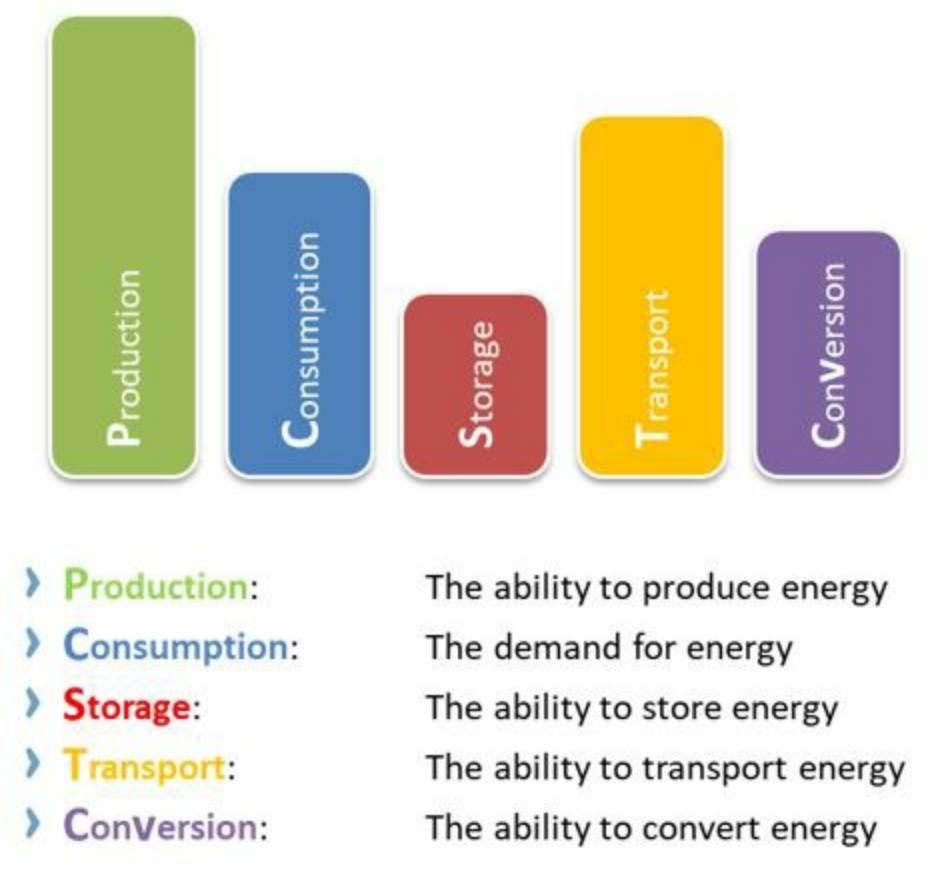
This chapter describes the ESDL concept. The following topics will be covered:

- Design principles: the general design principles behind the language: capabilities and aggregation
- Data types: the most important data types in the language: EnergySystem, Instance, Potential, Area, Asset, EnergyAsset, Port, Profile

# Design principles

## Energy Capabilities

There are vast amount of different assets in an energy system all having their specific characteristics (e.g cables, heat networks, power plants, solar PV and many more). Modelling an energy system in much detail therefore requires lots of work describing all different assets. However, if we look into the different assets and compare theirs behaviour we see that primarily all assets provide one (or more) core functionality and can all be categorized in one of the following five capabilities.




Some examples:

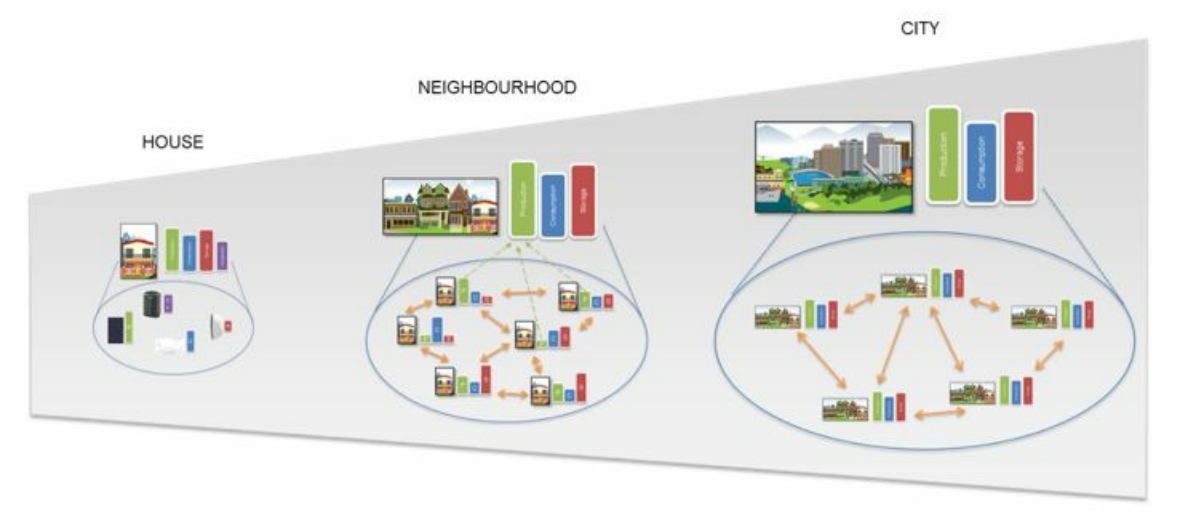
- Production: SolarPV panel, Wind Turbine, etc.
- Consumption: Households electricity consumption, heat consumption of a city, etc.
- Storage: Home battery, heat buffer, etc.
- Transport: District heating, electricity grid, etc.
- Conversion: Heat Pump, Transformer, Gas burner, etc.

Describing the assets in terms of capabilities achieves two things that are very welcome. Firstly we can make an abstraction of an assets by only modelling characteristics that are relevant per capability. This results in an simplification for the user of ESDL since he now can approach different asset types in a similar way. For instance a PV panel (Production) can be approached in a similar way as a wind turbine (Production) with slightly different characteristics. This makes life easier for those who want to reason on energy systems (evolution) with calculation tools.

Secondly, every capability comes with a natural way of aggregation. For instance, if the consumption capability for every house in a street is defined (for instance as electric power consumption/year), then the aggregated consumption capability of the whole street can be determined by calculating the sum of all the consumption capabilities. This is achieved by modelling all energy consumers as consumer capability. The following section describes the use of this aggregation in ESDL.

# Levels of aggregation

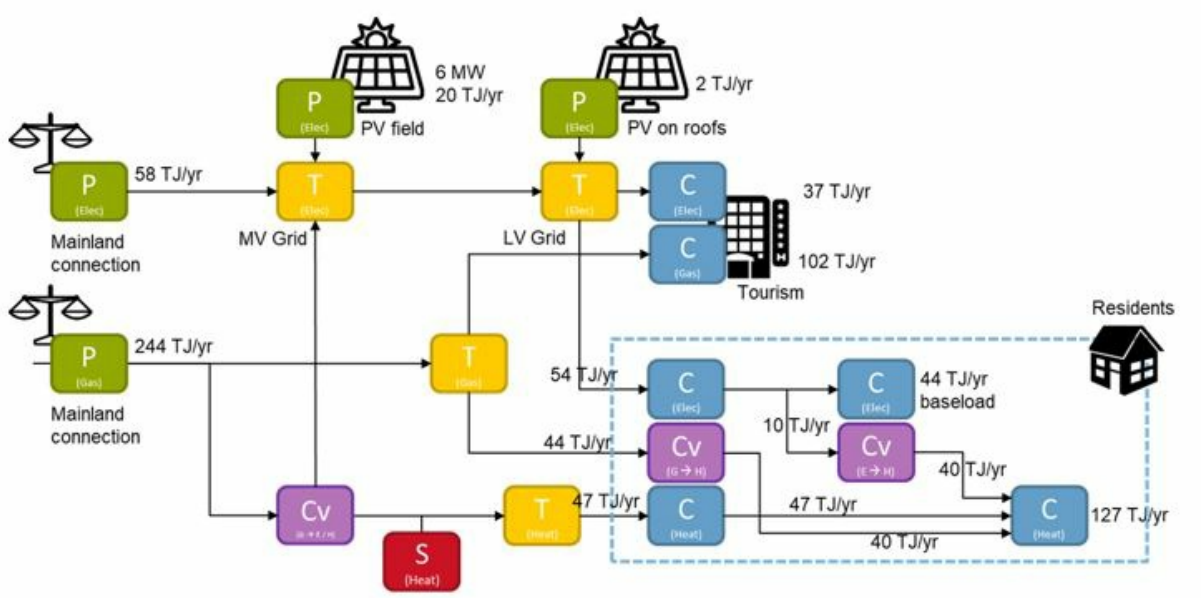
Modelling an energy system can be done in multiple different levels of detail (geographical scales). The level of detail needed depends on the application. For making a energy plan on country level it is much likely not necessary to model the energy consumption of every single building in that country separately with a high time resolution. However, for designing a district heating installation in a specific area it might be possible that that detailed information is necessary. ESDL does not prescribe a certain detail level of modelling of an energy system, it supports describing energy systems on multiple levels. Depending on the application, or the availability of data, a modeler can choose an appropriate modelling level




ESDL supports the modelling of energy systems on detail levels such as: house, building, neighbourhood, city etc . Because in ESDL everything is modelled in terms of capabilities (previous section) the different levels are also modelled in terms of capabilities. This means that for instance a house can have a certain level of production, consumption and storage, the ESDL model on neighbourhood scale can be expressed in the same capabilities by taking the aggregated values of the capabilities of all houses in that neighbourhood. The same can be done for city and country scales. This results in a generic way of modelling energy systems independent of the scale (detail level). This simplifies reasoning on different energy systems for applications.

# Example of modelling with energy capabilities

An model of a possible future energy system of the island of Ameland, a small island in the north of the Netherlands, is depicted in Figure 1.


6 MW
20 TJ/yr
PV field
2 TJ/yr
PV on roofs
58 TJ/yr
Mainland connection
MV Grid
LV Grid
37 TJ/yr
102 TJ/yr
Tourism
Residents
244 TJ/yr
Mainland connection
54 TJ/yr
44 TJ/yr baseload
10 TJ/yr
44 TJ/yr
40 TJ/yr
47 TJ/yr
47 TJ/yr
127 TJ/yr
40 TJ/yr

# Data types

ESDL contains several first-class data types that are used in each ESDL-file. This section describes these data types.

## EnergySystem

This is the entry point of an ESDL-file: every ESDL-file starts with a definition of an Energy System. EnergySystem. An Energy System contain Instances, EnergySystemInformation, Measures, Parties and Potentials.

## Area

An area allows for grouping Assets and Buildings and other Area's. Since real Energy Systems have a physical representation, Area scope the given Energy System by this physical boundary. Every Energy System describes at least one Area.
Area

## Items, Assets, EnergyAssets and Services

While Items represent logical things (both Assets and Services) in an Energy System, Assets represent physical things of an Energy System. Assets can be specialized into Buildings and in EnergyAssets. The difference between an EnergyAsset and an Item or Building is that Energy Assets contain ports and these ports allow them to connect to other EnergyAssets, allowing to create a network (graph) of how the energy system is connected. EnergyAssets themselfs are specialized into the five ESDL categories: Producer, Consumer, Storage, Transport and Conversion.

Items, Assets, EnergyAssets and Services
Overview of EnergyAssets

## Profiles

Profiles are used to model values in ESDL. ESDL suports different types of values, such as single values, time ranges, time series, etd. E.g. the total consumption of energy in a municipality is 50PJ. The price of a HeatPump is 8000 euro. These values can be stored in a profile called 'SingleValue' as they contain only one value.
In many applications, e.g. simulations, ranges of values are required as input, that are mostly time indexed. These timeseries-based values can be stored in DateTimeProfiles.
Profiles are used to define information about the energy system as a whole (e.g. the price-index of 2017, APEX energy prices of 2015-2017). The Energy System Information part of an EnergySystem can be used for that. Profiles can also be used for specifying e.g. the electricity demand of a household. For that a profile with this demand can be attached to a port of the ElectricityDemand class.

- Profiles

# Energy System

The EnergySystem class will be the place to start if you want to model an energy system for a certain region.

An EnergySystem contains the following information:

- Instances
- Potentials
- Measures
- Parties
- EnergySystemInformation

# Instances

An EnergySystem can contain zero or more Instances. Instances are used to represent different representations of the **same** EnergySystem. Most of the times only one Instance will be used. The primary use case for having more than one Instance is when you have different aggregations of the same EnergySystem in the same model (e.g. the same region on house level and aggregated on neighbourhood level). Another option would be to create different instances for different years (to describe the progress of the energy transition).

All instances contain an Area. The Area is the class to represent a geographical or logical area. An Area contains all assets in that area or can be subdivided into more Areas

# Potentials

Potentials can be used to describe all different kinds of energy potentials. Examples of potentials are:

- GeothermalPotential: to describe the potential for geothermal energy
- WindPotential: to describe the potential for wind turbines
- SolarPotential: to describe the potential for solar PV fields
- LegalArea: to describe for example areas where no aquifers are allowed because the underground water is used for drinking water (a negative potential).

# Measures

Measures can be used to describe possible assets that can be installed in the EnergySystem (e.g. added by a model calculating minimum investments required to match future demands).

# Parties

Parties can be used if you want to model ownership of assets

# EnergySystemInformation

The EnergySystemInformation class is a container for additional information about the EnergySystem in general.

Currently the following information can be added:

- AvailableEnergyCarriers: list of available energy carriers with information about energy content and CO2 emissions. Power plants typically refer to energy carriers to indicate what the source of their energy is.
- EnergyPrices: prices for gas, electricity

# Areas

The Area class represents a physical geographic area or a more abstract logical area. In both cases it is the 'asset container', in a sense that all assets within the area are contained by the Area instance. A physical geographic area is a specific area with a location and geometry (and can be pointed at on a map). A logical area can be an non-specific area, like 'a neighbourhood with houses from the period 1970-1990' without specifically stating where this neighbourhood is located.

An area can have a scope (like country, region, city, village, street, building and lots more) and have a type (like land, road, water, sea, ...). An area can be subdivided into smaller areas, like a city that is divided into neighbourhoods and neighbourhoods being divided into streets.

Areas can have the following properties:

- Location and geometry
- EnergyPotential (wind, solar PV, geothermal, aquifer, ...)
- Legal restrictions
- Social properties
- Economic properties

# Items, Assets, EnergyAssets and Services

## Items

The Item class represents an abstract item. Items can have an id, a name, a short name and a description. The following classes are derived from the item class:

- Assets: an Asset is an Item with a physical representation.
- Services: A Service is a logical Item.

## Assets

Assets are all physical items in the EnergySystem. Assets can have a location, a geometry, commissioning and decommissioning dates, cost information (investment, installation and operation and maintenance costs).

Assets are contained by Areas.

## EnergyAssets

An EnergyAsset is an Asset with one or more ports allowing to model connections between assets.

The EnergyAsset class is subdivided into 5 categories:

- ProductionAsset: e.g. wind turbine, solar panel, geothermal source
- ConsumptionAsset: e.g. electricity demand, heat demand, gas demand
- StorageAsset: e.g. battery, buffer
- TransportAsset: e.g. electricity network, gas network, cable, pipe, transformer, heat exchange
- ConversionAsset: e.g. gas heater, heat pump, power plant, CHP, fuel cell

## Services

Example Services are demand response services and aggregator services.

# Overview of EnergyAssets

All asset classes have a Generic and a Aggregated version, e.g. GenericProducer and AggregatedProducer. The Generic version can be used if no specific class is available or required. This can only be used when no specific parameters are required. In some examples only the presence of the asset is enough information. The Aggregated version can be used when the asset is a reprensentation of more than one physical assets. Aggregated assets also allow to specify the links to the individual assets it represents (For this purpose more instances within one energysystem can be used, one instance with the individual assets and a second instance with the aggregated versions).

# Producer Assets

Currently the following producer assets exist:

- PVPanel: Represents an individual PV panel or an installation of PV panels on a roof for example.
- PVParc: Represents a PV parc.
- SolarCollector: Represents an individual solar collector or an installation of solar collector panels.
- WindTurbine: Represents an individual windturbinbe.
- WindParc: Represents a Wind parc (on sea or on land).
- GeothermalSource: Represents a geothermal source producing heat or electricity.
- SourceProducer: Represents an unlimited source of energy, that can be used to represent the network outside the scope of the energysystem being described.
- ResidualHeatSource: Represents residual heat sources from data centers, industry or cooling houses.

New producer assets will be added in future.

# Consumer Assets

Currently the following consumer assets exist:

- ElectricityDemand: Represents the electricity demand of a house, group of buildings, sector, area or country.
- GasDemand: Represents the gas demand of a house, group of buildings, sector, area or country.
- HeatingDemand: Represents the heating demand of a house, group of buildings, sector, area or country.
- CoolingDemand: Represents the cooling demand of a house, group of buildings, sector, area or country.
- EnergyDemand: Represents the energy demand of a house, group of buildings, sector, area or country. It can be used in describing the energybalance of a region for example.
- Losses: Represents the losses in an energy system.
- EVChargingStation: Represents the electricity consumption of an EV charging station.
- SinkConsumer: Represents an unlimited sink of energy, that can be used to represent the network outside the scope of the energysystem being described.
- MobilityDemand: Represents the energy demand of the mobility sector. Can be used to represent total demand or specify a demand per vehicle type and/or fuel type.

New consumer assets will be added in future.

# Storage Assets

Currently the following storage assets exist:

- Battery: Represents an electrical battery to store electricity.
- HeatStorage: Represents a water buffer to store heat (or cold).

New storage assets will be added in future (aquifer storage).

# Transport Assets

Currently the following transport assets exist:

- ElectricityNetwork: Represents an electricity network (on any scale, from an inhouse network, to the national scale electricity network.
- EConnection: Represents the connection to a electricity network (including the electricity meter).
- ElectricityCable: Represents an individual cable in a network (if you want to describe a detailed topology).
- Transformer: Represents a electric transformer (station) to connect two networks with different voltage levels.
- HeatNetwork: Represents a (district) heat network.
- HConnection: Represents the connection to a heat network.
- HeatExchange: Represents a heat exchange to connect two heat networks.
- HeatPipe: Represents an indivual pipe (if you want to describe a detailed topology).
- Pump: Represents a pump to generate a flow of water in a pipe or network.
- Valve: Represents a valve to control water flow.
- GasNetwork: Represents a gas network.
- GConnection: Represents the connection to a gas network (including the gas meter).
- EnergyNetwork: Represents an (abstract) energy network (for describing energy balance).

New transport assets will be added in future.

# Conversion Assets

Currently the following converion assets exist:

- GasHeater: Represents a gas heater.
- HeatPump: Represents a heat pump (e.g. air-water, connected to an aquifer, using surface water, as a booster pump)
- CHP: Represents a CHP (Combined Heat and Power) to generate electricity and useful heat simultaneously.
- FuelCell: Represents a fuel cell, that generates electricity (and possibly heat) using a certain fuel (H2 for example).
- PowerPlant: Represents a power plant, using for example coal, gas or uranium to produce electricity (residual heat can be delieverd to a heat network).
- Airco: Represents an airconditioner.
- FermentationPlant: Represents a fermentation plant.

New conversion assets will be added in future.

# Profiles

ESDL support many different types of profiles. Next to this ESDL uses profiles for different applications.

Currently ESDL uses profiles for:

- describing energy consumption and production at ports of an `EnergyAsset`
- describing solar irradiance or wind speeds over time
- describing historic, current and expexted costs for an `EnergyCarrier` or `Commodity` (e.g. expected electricity price developments)
- describing costs for an `EnergyAsset` (cost developments of a HeatPump for the coming 30 years). These costs are divided into investment costs, installation costs and operation and maintenance costs)

ESDL supports the following types of profiles:

- `DateTimeProfile` : Allows to specify a range of values with timestamps.
- `SingleValue` : Allows to specify a single value for a parameter
- `Range` : Allows to specify a range of possible values between minValue and maxValue.
- `URIProfile` : Allows to refer to a URI for a profile
- `InfluxDBProfile` : Allows to refer to a profile in an InfluxDB database
- `ReferenceProfile` : Allows to refer to a profile in the `Profile` collection class.

# DateTimeProfile

The `DateTimeProfile` is a collection of values each annotated with a start time ( `from` ) and optionally an end time ( `to` ). When only a starting time is specified, the value is valid until the start time of the next value in the profile. So, the value of the last item in the profile, is valid for the rest of the time (basically forever). If you want to specify a value for a certain bounded time period, you must specify the end time.

# InfluxDBProfile

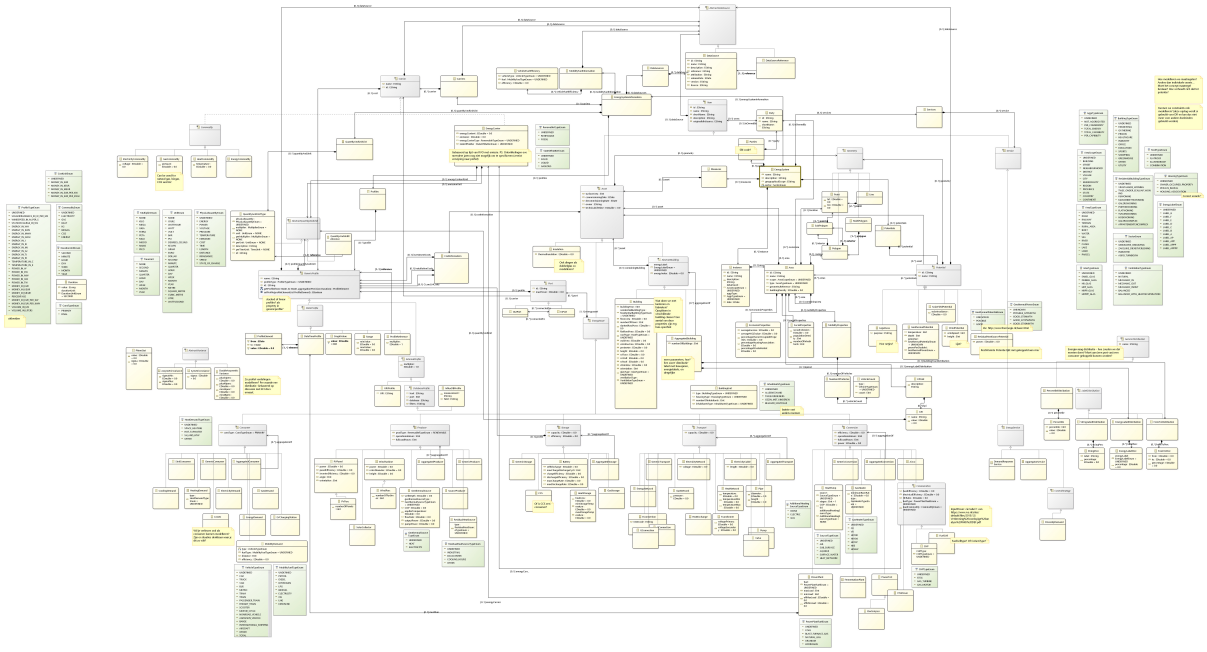
The `InfluxDBProfile` allows to refer to a profile in an InfluxDB database.

It requires the following parameters:

- `host` : the hostname of the database server
- `port` : the port on which the InfluxDB server is running
- `database` : the name of the database that contains the profiles
- `measurement` : the name of the measurement that contains the profile data
- `field` : the name of the field that contains the profile values

# ESDL model

A graphical view on the ESDL model can be found here:



Source: https://raw.githubusercontent.com/EnergyTransition/ESDL/master/esdl/model/esdl.png

# How to use ESDL

The possible applications of ESDL are endless. This chapter gives some guidelines of how to use the ESDL language.

# Models and descriptions (TODO)

...

# Different levels of abstraction and scopes

ESDL is a very generic energy modelling language. It can be used to describe energy related systems with very different abstration levels or scopes:

- describe an individual house or a street of houses with their interconnection. For each house the electricity, heating and/or cooling demand can be described and the present assets (heatpump, gasheater, in house battery, rooftop PV, EV charging stations). Also different house parameters can be described (insulation parameters, energy label, areas of walls/windows/roofs, type of roof (slanted or flat roof)).
- describe a city with different districts, where buildings are clustered as aggregated buildings. Each district has its own characteristics (type of houses, building periods) and possible renewable energy solutions. Some districts are suited for a district heating network solution, others are suited for an all-electric approach.
- describe a region with the energy demand of different sectors (e.g. industry, agriculture, residential, businesses), including the installed production assets, and available renewable production and energy storage options.
- describe the energy balance of a country, indicating imports and exports of energy.
- describe the electricy production facilities in the country including the flow of 'supply materials' (coal, gas, oil).

It can also be used to describe all kinds of energy related (open) data, either to publish data for use by others or to document assumptions taken into account in a model or calculation. The following list provides some example applications:

- standard energy profiles (with consumption levels for every 5 minutes).
- CO2 emission data for different energy carriers.
- historic, current or predicted fuel prices.
- yearly profile for solar irradiance (or wind speeds) for a certain region.
- data sheets for different components in the energysystem (with specific parameter values).
- expected prices for energy assets for the coming 30 years (what will a heat pump cost in 2040).
- geographic data: potential geothermal energy in the underground, total electricity consumption per municipality for a whole country.
- distribution of energy labels over all houses in the city.

# Geographic and non-geographic models

ESDL can be used for geographic models and non-geographic models. Geographic models contain physical shapes and locations for all assets in the energysystem. It will mostly be used for describing a specific actual existing region and its future (or historic) developments. Non-geographic models do not contain locations and can

be used to describe some commonly available configurations of energysystems without denoting a specific one, e.g. city centers with mainly old houses or an energy neutral house that is fully self sufficient.

# Start with (TODO):

Depending on the application, you can start your ESDL model with:

- EnergySystem
- Carriers
- Profiles
- House
- an Asset template

# Tooling for ESDL

## Modelling

The concepts and the relations among the concepts in ESDL are defined in a structured model: the ESDL model. A modelling language is subsequently used to model the ESDL model. This modelling language is called the Unified Modelling Language (UML). But to make ESDL more accessible to both modelers, developers and end-users, ESDL is based on a dialect of UML called ECore. Therefore ESDL is modelled in ECore.

ECore is part of the Eclipse Modelling Framework (EMF). Eclipse is a well-known open-source Integrated Development Environment (IDE) that supports developers with all kinds of tools to develop software programs.

The EMF gives us several tools for free compared to using UML:

- Code generation. It can automatically generate Java-classes, XSD schema's and UML models from the ECore model.
- Provide semantics for modelling concepts. These semantics are geared towards generating tools to read, visualize and edit your ESDL files. EMF provides two ways for these semantics: (1) a static version (using the code generation facility, creating actual files on disk), and (2) a dynamic version that is in real-time created from the concepts you are modelling. This allows for direct feedback on the changes you make to the model as editors are instantly updated with these changes.
- A graphical editor to manipulate the model.

These properties make it easier to adopt ESDL, as this project not only provides an XML Schema for usage in energy transitions tools, letting the developers write XML-based ESDL-files themselves, but also provide rich editors to manipulate ESDL-files.

The tools can be spilt up in two categories:

1. Tools to edit the ESDL model instances (the ESDL-files)
2. Tools to edit the ESDL model itself and contribute the ESDL specification (the ecore-model)

# Using ESDL to model an energy system

## Setting up Eclipse

You can install Eclipse using the link Setup Eclipse using the update site. You will get automatic notifications whenever updates become available.

## Using ESDL to model an energy system

At the moment there are two initiatives to view and edit/generate ESDL instances:

- Eclipse based ESDL tree editor
- Graphical ESDL editor (the designer)

# Setup Eclipse using the update site

# Installing Eclipse ESDL editor

## Step 0: Download Oracle JDK (if necessary)

Go to www.oracle.com/technetwork/java/javase/downloads/index.html and press the “Download” button under the “JDK” text to download the JDK. Version 1.8 or higher should be fine.

## Step 1: Download Eclipse Modelling Tools

From: http://www.eclipse.org/downloads/packages/

Go to the section "Eclipse Modelling Tools" and select for example “Windows 64-bit”: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/photon/R/eclipse-modeling-photon-R-win32-x86_64.zip

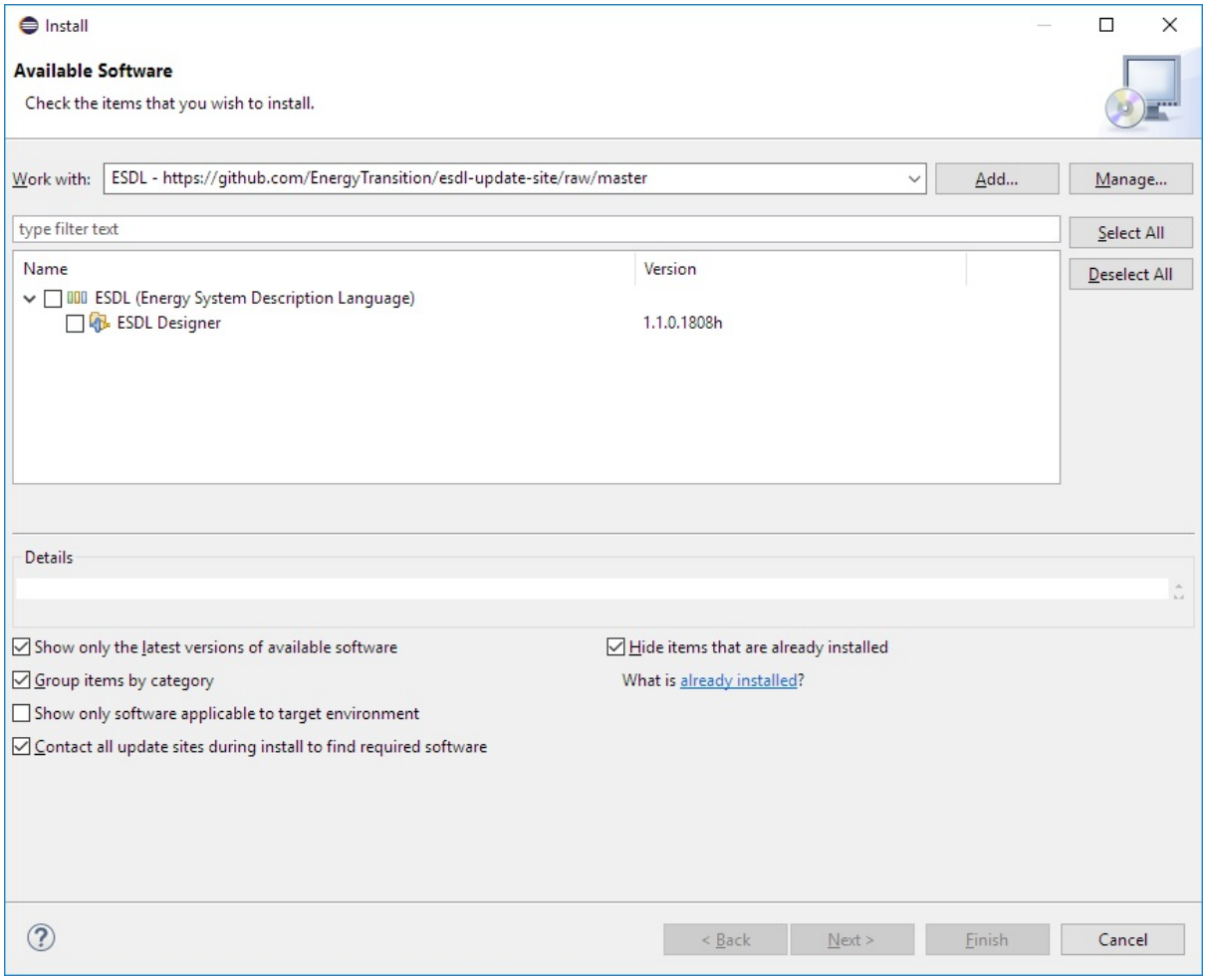
# Add ESDL plugin update site

To install all the required components and edit ESDL-files, an Eclipse update-site is available to get the latest version of the ESDL Designer and the ESDL model.

To do that, you need to add an eclipse update site to your eclipse installation.

## Step 1: Open eclipse and close "Welcome Screen"

## Step 2: Select "Help" --> "Install new software..."




## Step 3: Press "Add..." to add a new software repository

Give the Repository a name and use the following URL as location:

``` `https://github.com/EnergyTransition/esdl-update-site/raw/master`

You can also drag and drop the above URL on the "Install new software" dialog (depicted above) or the "Add new repository" dialog.

## Step 4: Select the package "ESDL (Energy System Description Language)"

Follow the required steps to let Eclipse add the required ESDL components.

Eclipse in now configured to use ESDL and you will receive notifications whenever there are updates.
```

# ESDL Tree editor

## Introduction

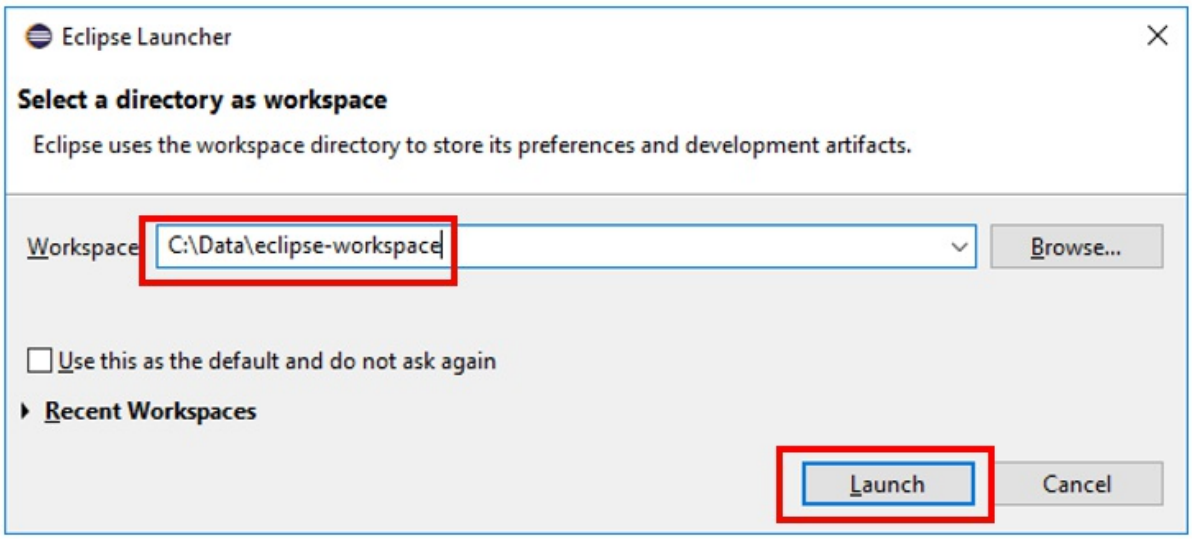
This document describes defining your first ESDL model using the ESDL tree editor. The Tree editor shows ESDL-files as an hierarchical tree and is directly generated from the ecore-model that defines ESDL. Therefore all concepts defined in ESDL are automatically available in the Tree editor.

## Defining your first ESDL model

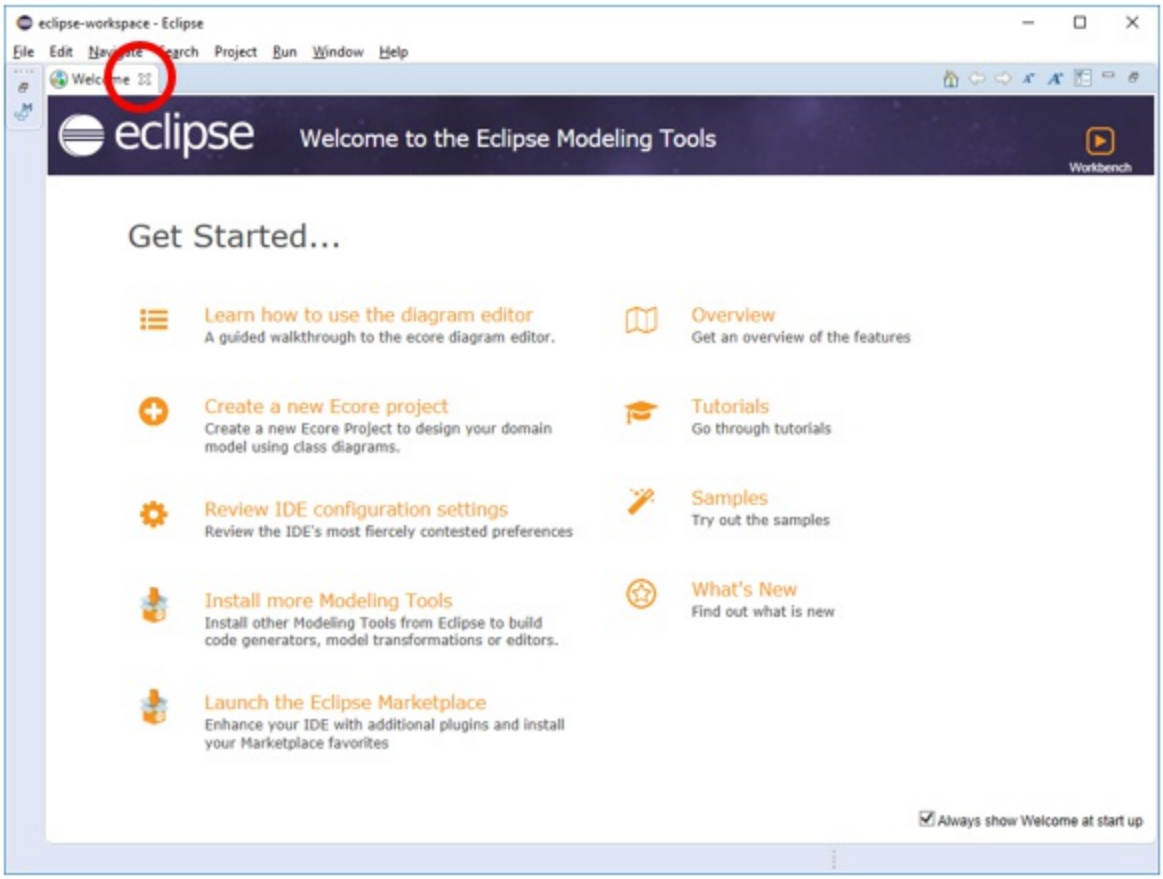


### Step 1: Start Eclipse and select workspace

Start eclipse by executing eclipse.exe from the folder your installed eclipse in Select a directory to store all your eclipse projects and press the “Launch” button:

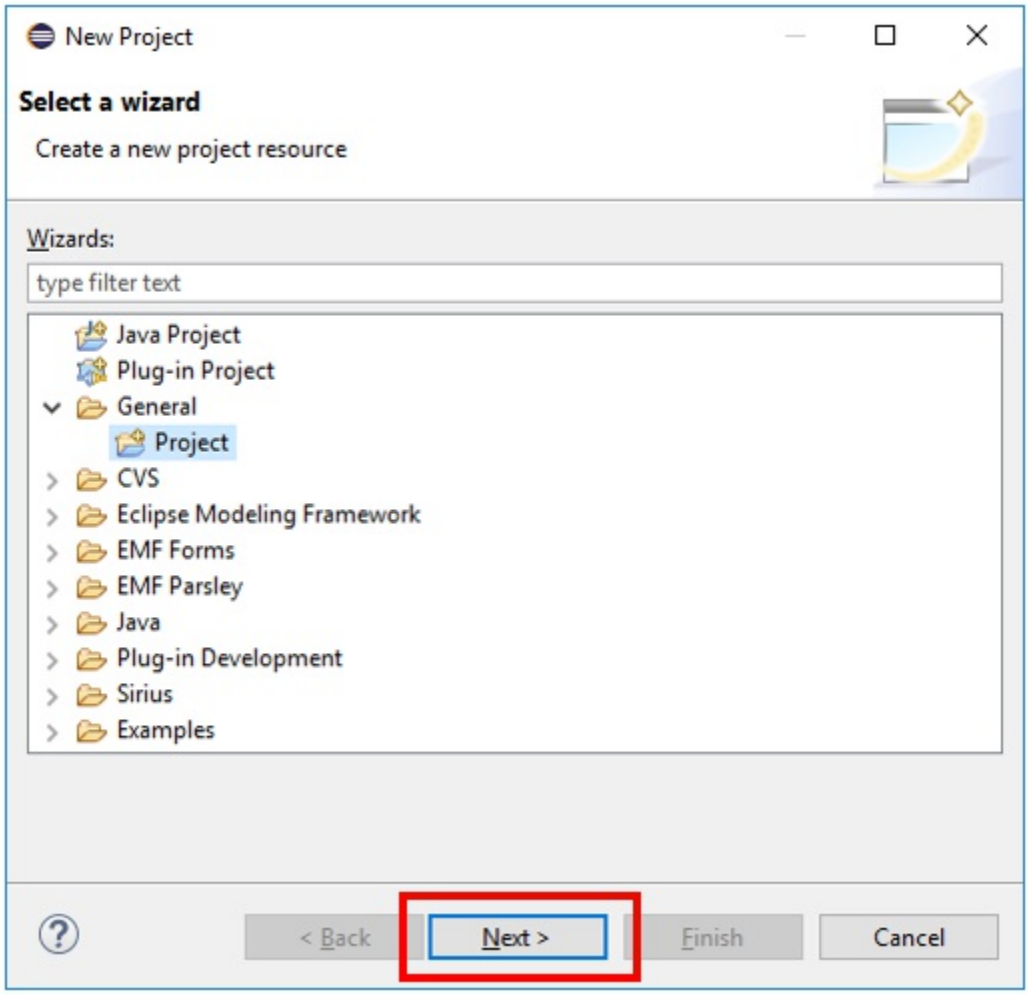



Press the “X” in the welcome window:

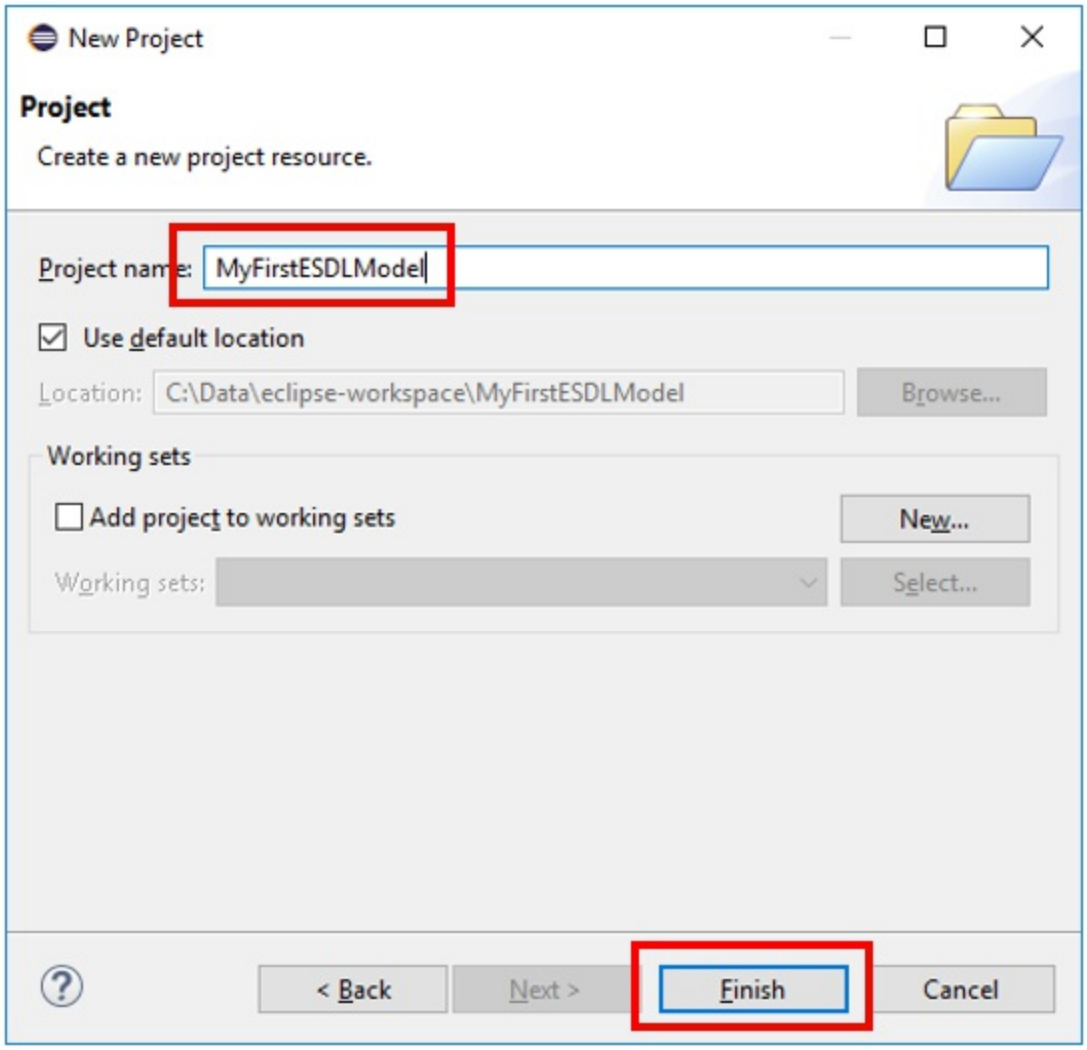



## Step 2: Create new project

Select “File” --> “New” --> “Project…” Select “Project” from the “General” folder and click “Next”

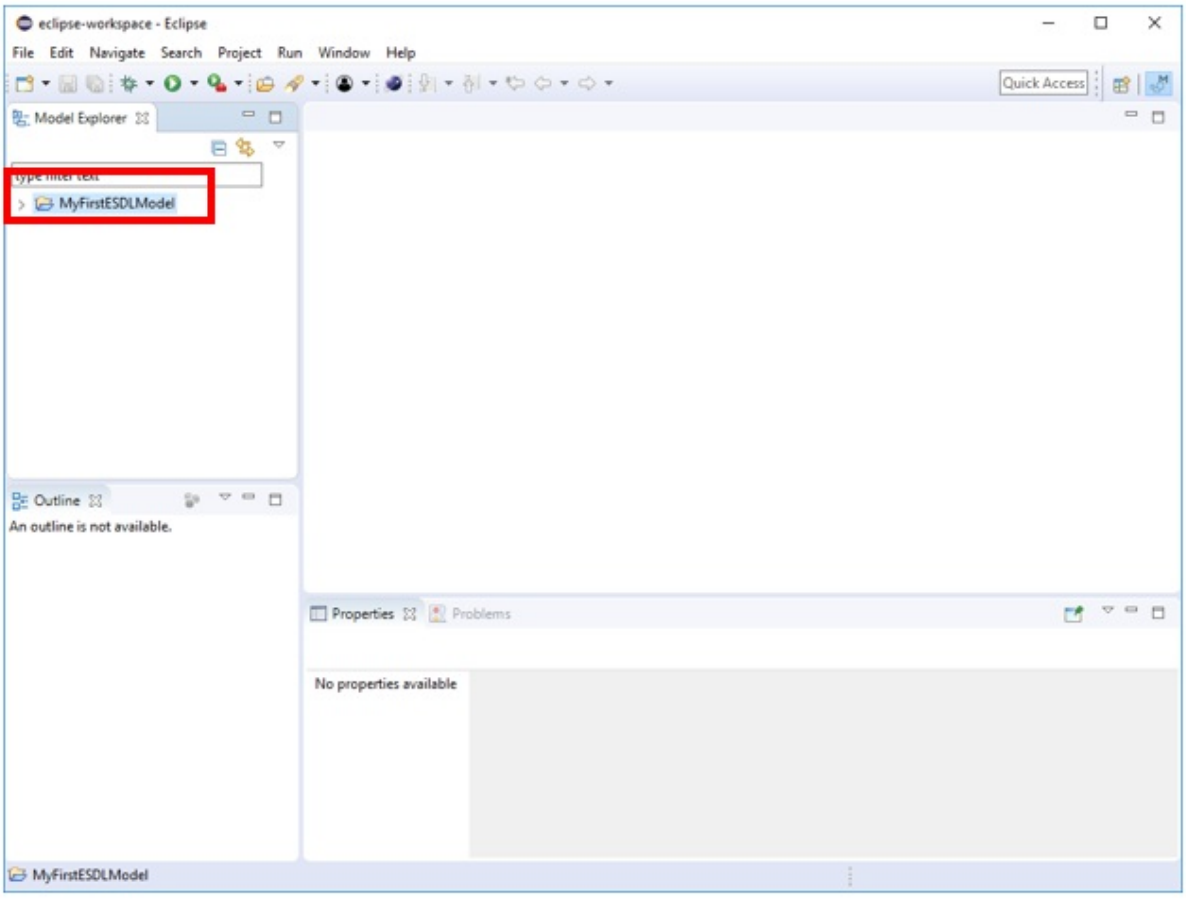



Give your project a name and click “Finish”:

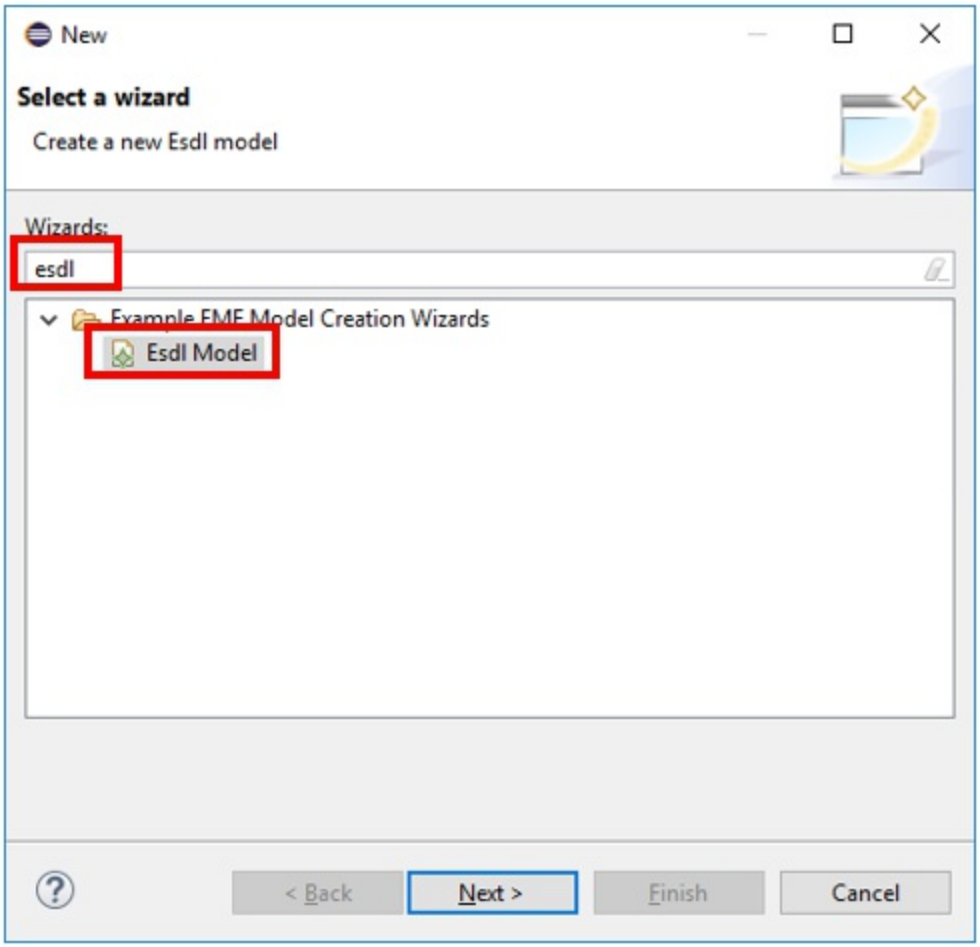



## Step 3: Create new ESDL model within the project

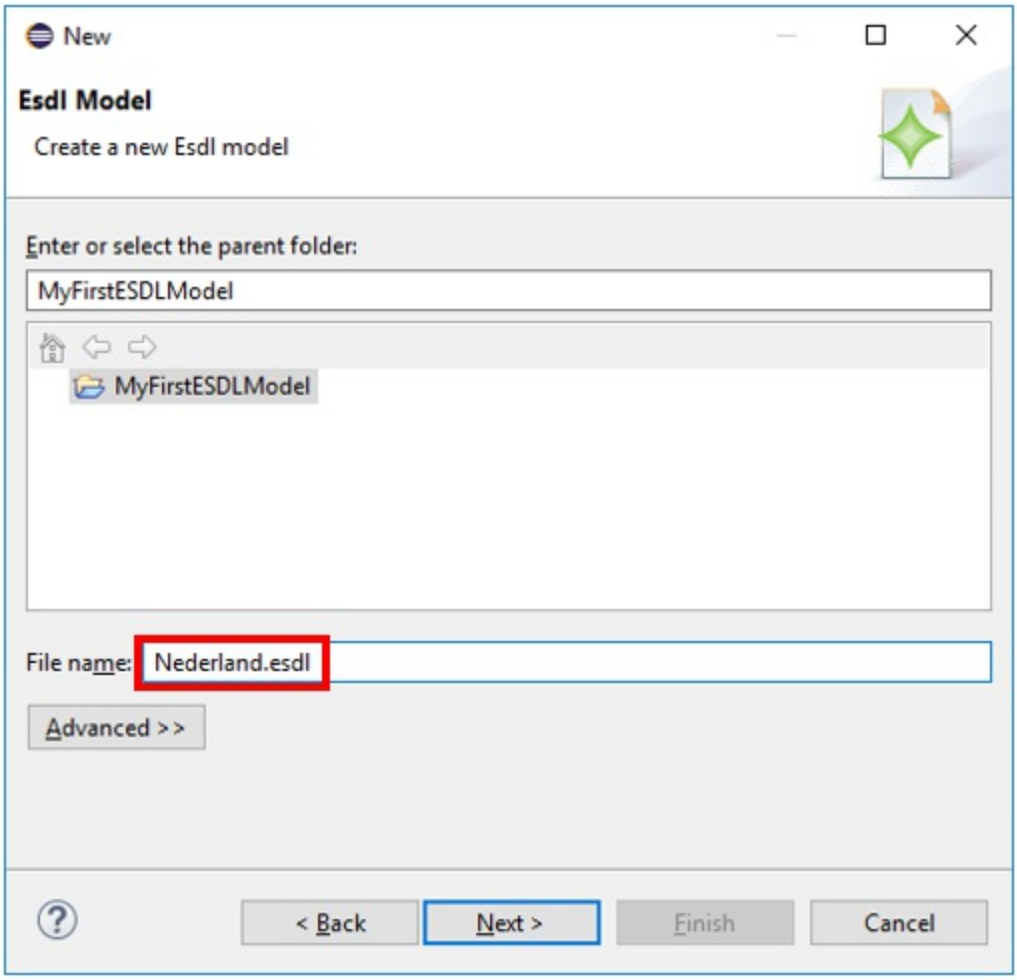
Right click on your project and select “New” --> “Other…”



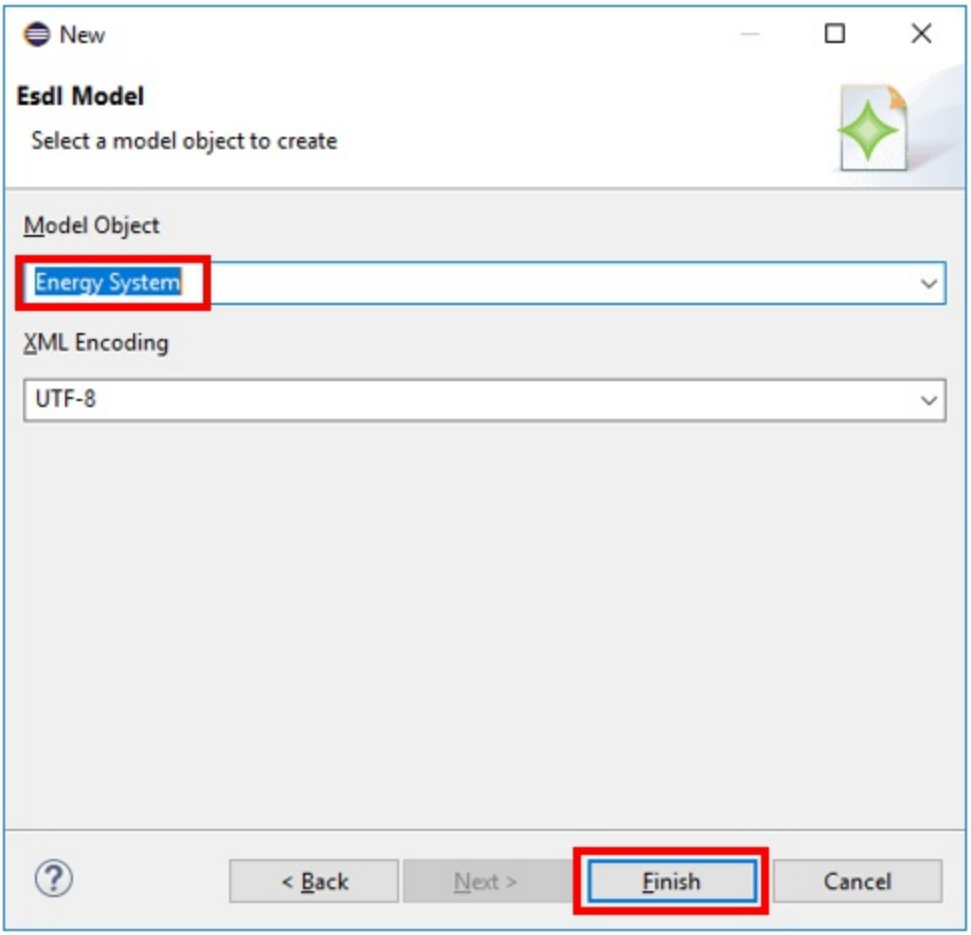

Enter “esdl” in the filter textbox and select “ESDL Model” from the available wizards:



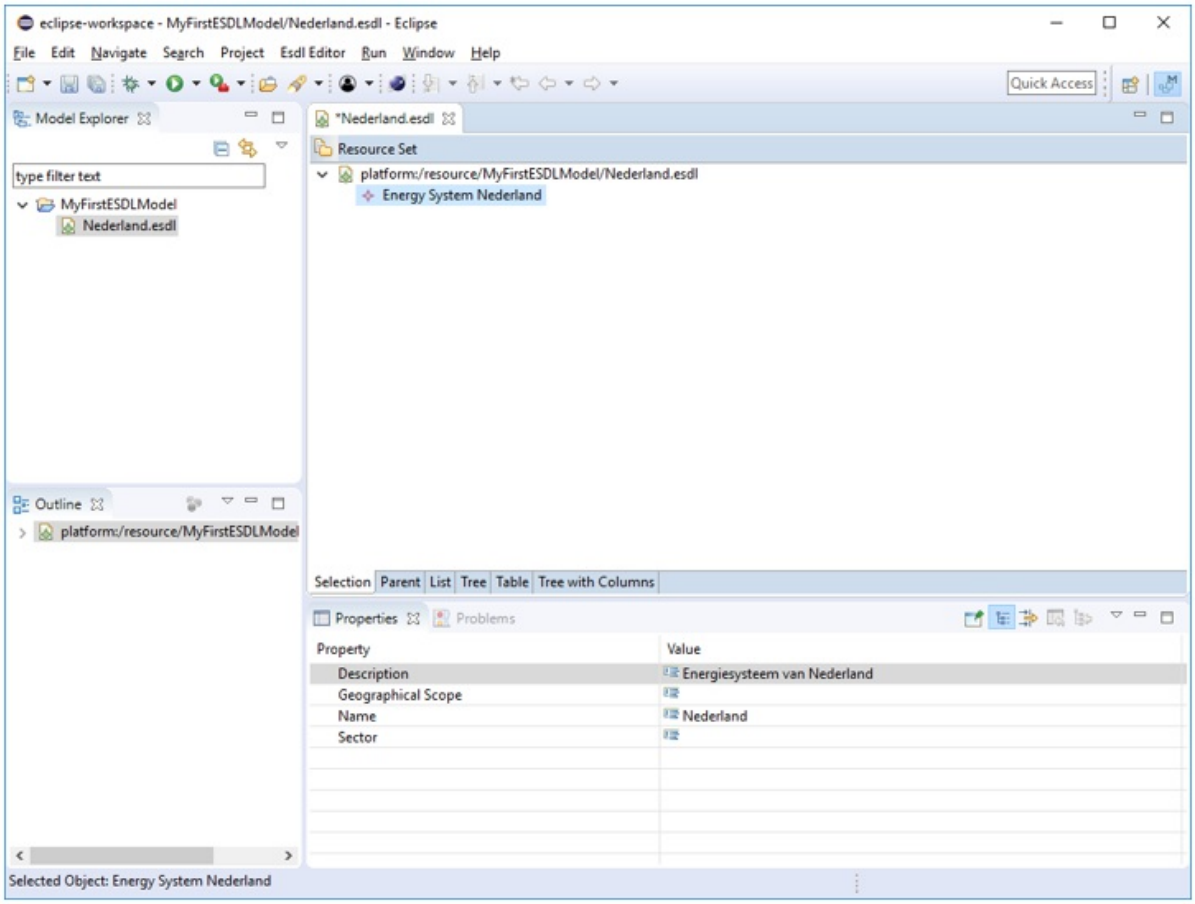

Give your ESDL model a name:




Select “Energy System” as a model object to create and click “Finish”:

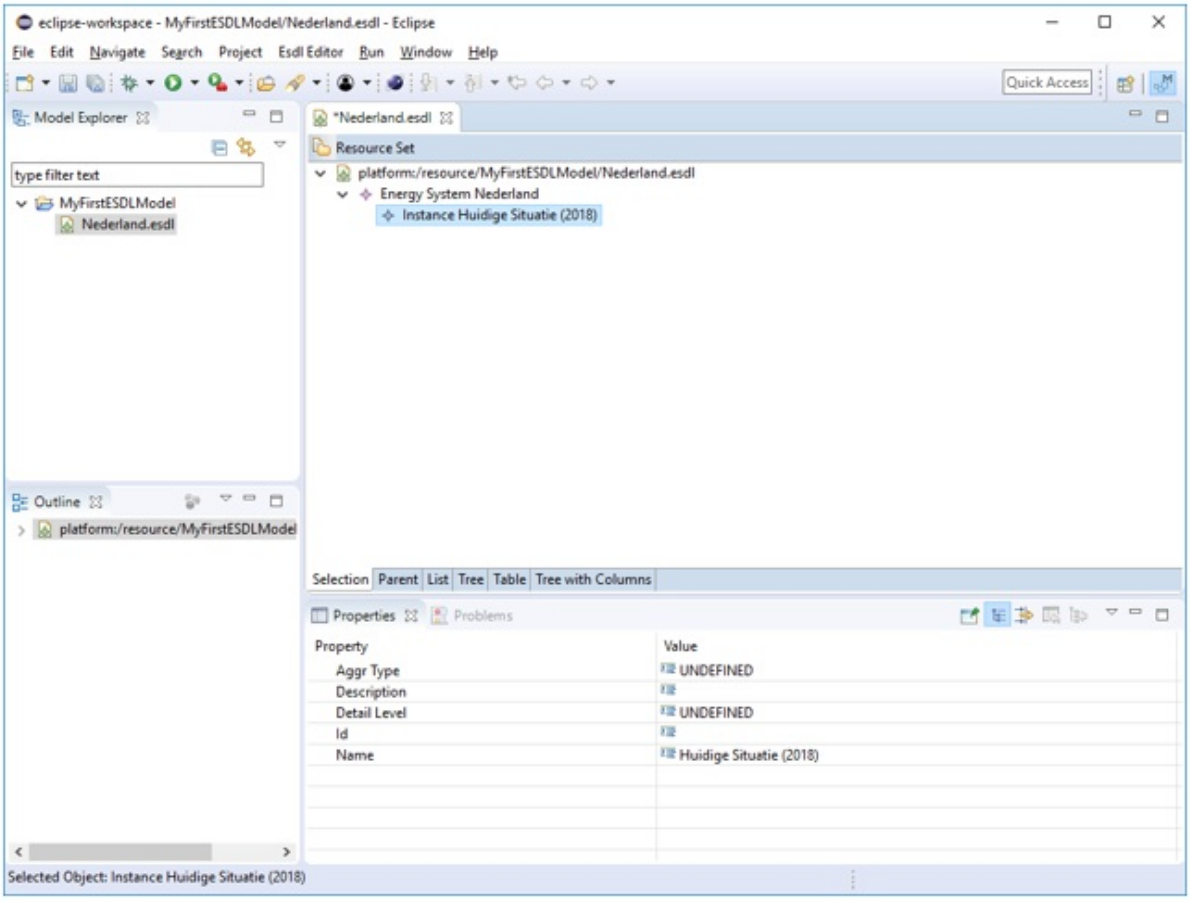



This opens the ESDL Tree editor for your newly generated ESDL model. The top part of the screen shows the newly created model in the Tree editor. In the bottom of the screen it shows the Properties of the currently selected ESDL-concept and allows you to manipulate these properties.:

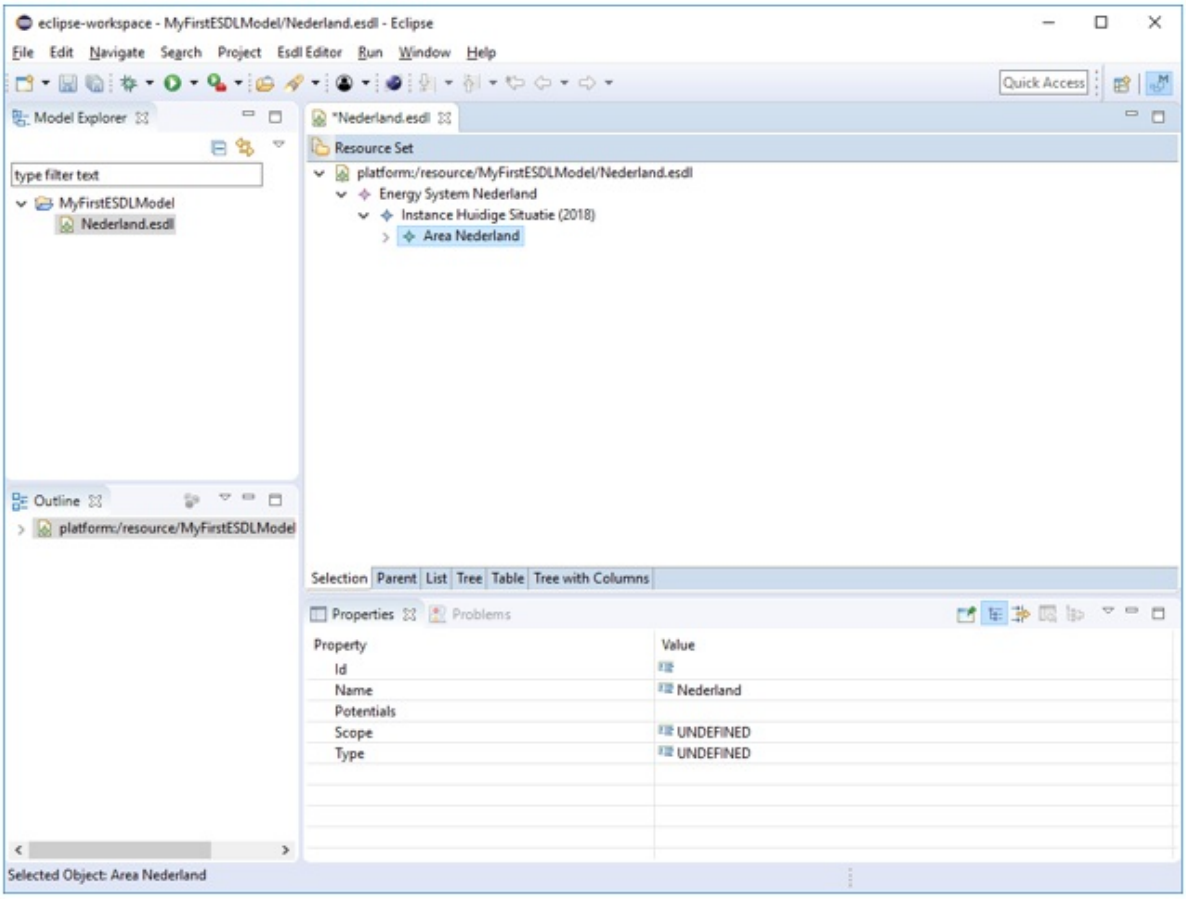



## Step 4: Edit ESDL model

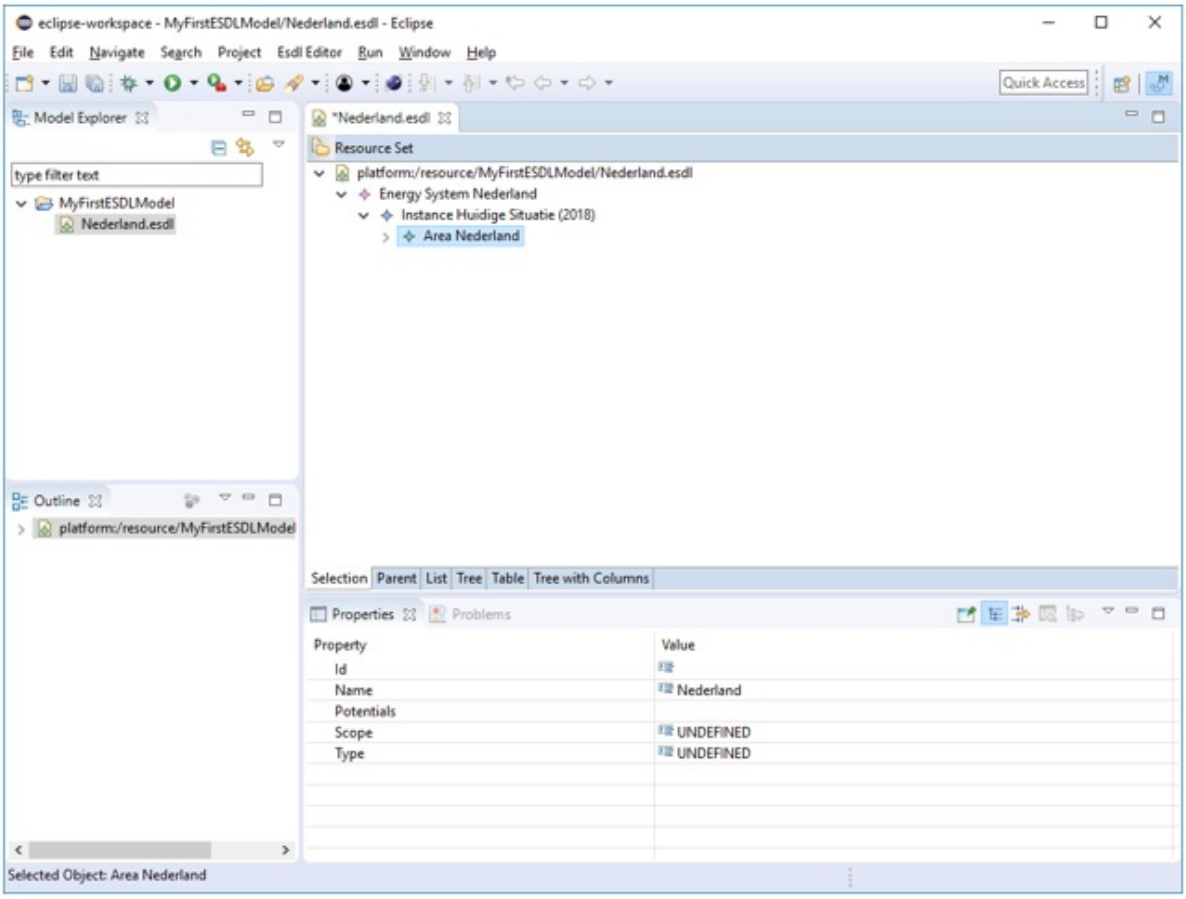
Right click on the Energy System and select “New Child” --> “Instance” and give the instance a name in the properties panel:




Add an area to the instance and give the area a name:




Add more areas or energy assets according to your needs… Start playing around!


eclipse-workspace - MyFirstESDLModel/Nederland.esdl - Eclipse
File Edit Navigate Search Project Esdl Editor Run Window Help
Quick Access
Model Explorer
type filter text
MyFirstESDLModel
Nederland.esdl
*Nederland.esdl
Resource Set
platform:/resource/MyFirstESDLModel/Nederland.esdl
Energy System Nederland
Instance Huidige Situatie (2018)
Area Nederland
Outline
platform:/resource/MyFirstESDLModel
Selection Parent List Tree Table Tree with Columns
Properties
Problems
Property
Value
Id
Name
Nederland
Potentials
Scope
UNDEFINED
Type
UNDEFINED
Selected Object: Area Nederland

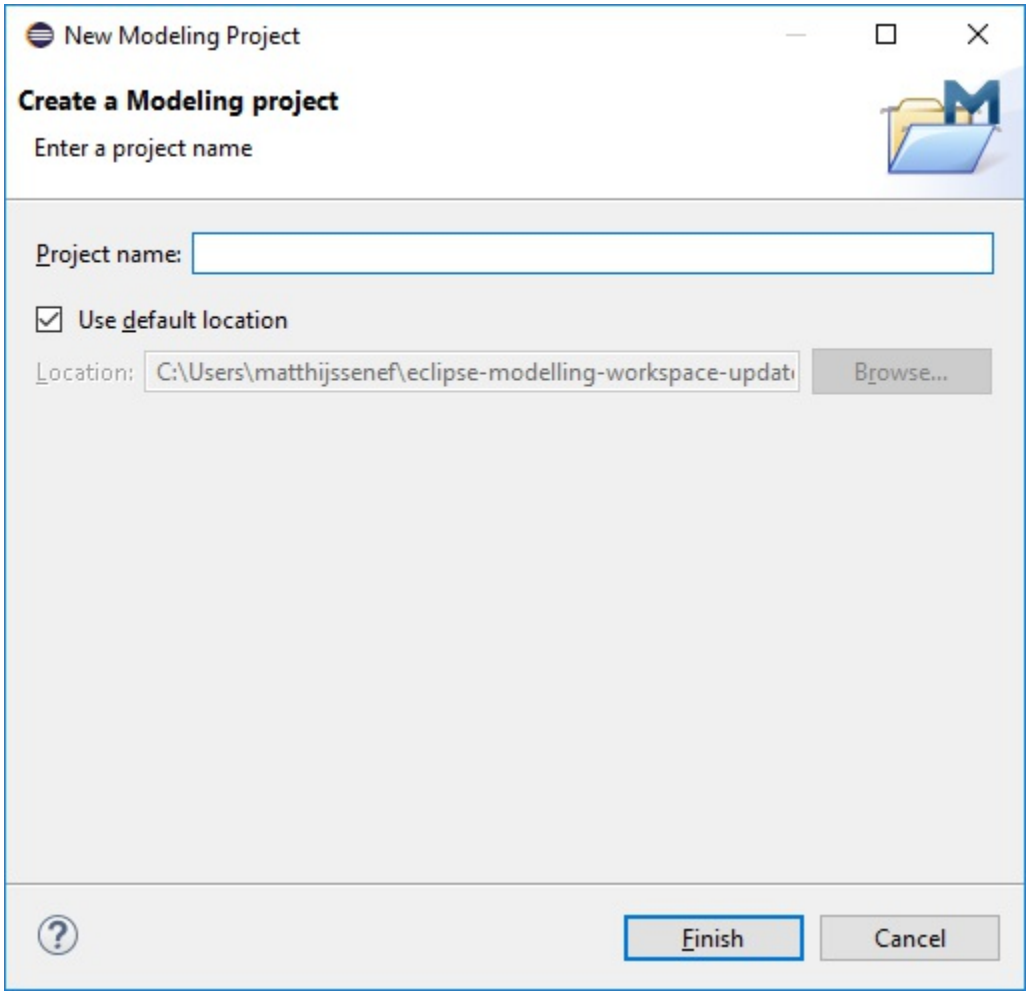
# ESDL Graphical editor / ESDL Designer

After installing the ESDL Eclipse plugins using the Eclipse update-site, you can start designing a new Energy System using this tutorial.

## Step 1. Create a new Modeling Project

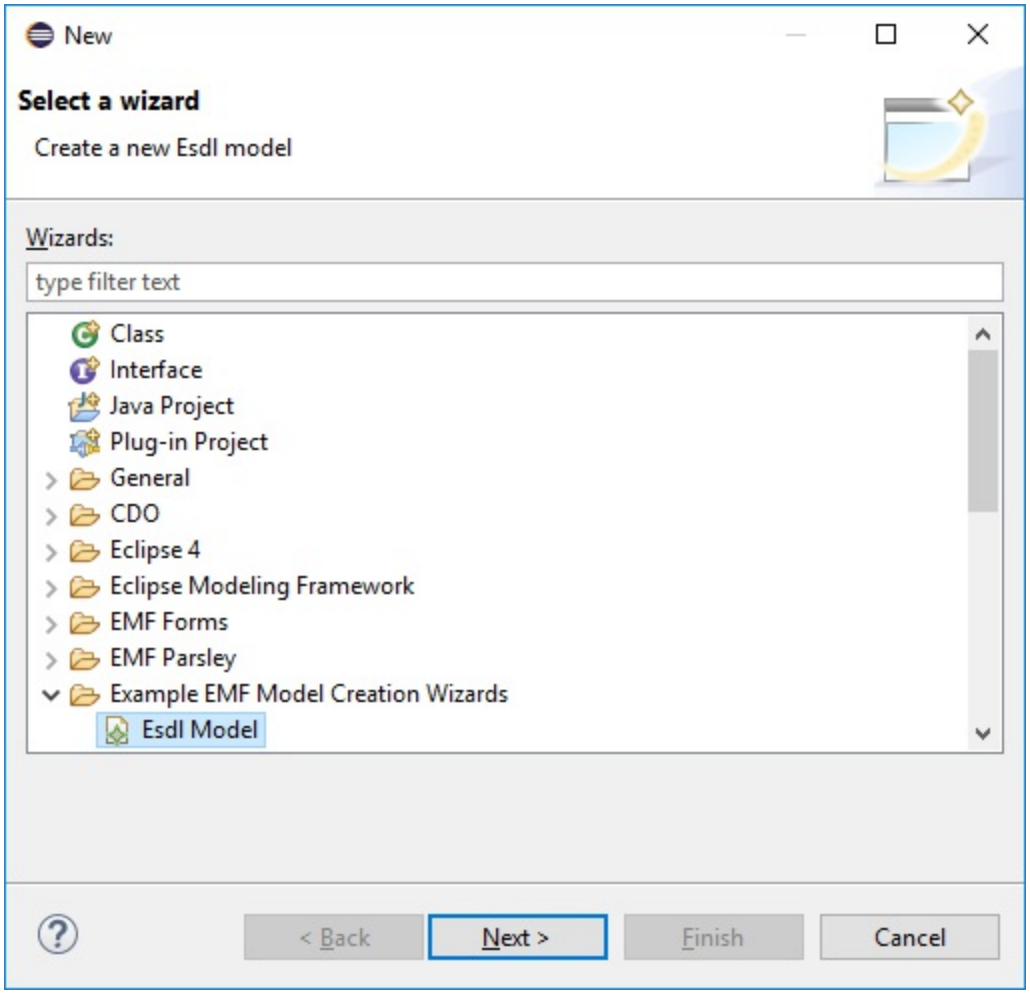
Select File --> New --> Modeling Project

Make sure you select a 'Modeling Project' since that will automatically create the required `representations.aird` file that records the layout of your diagram.



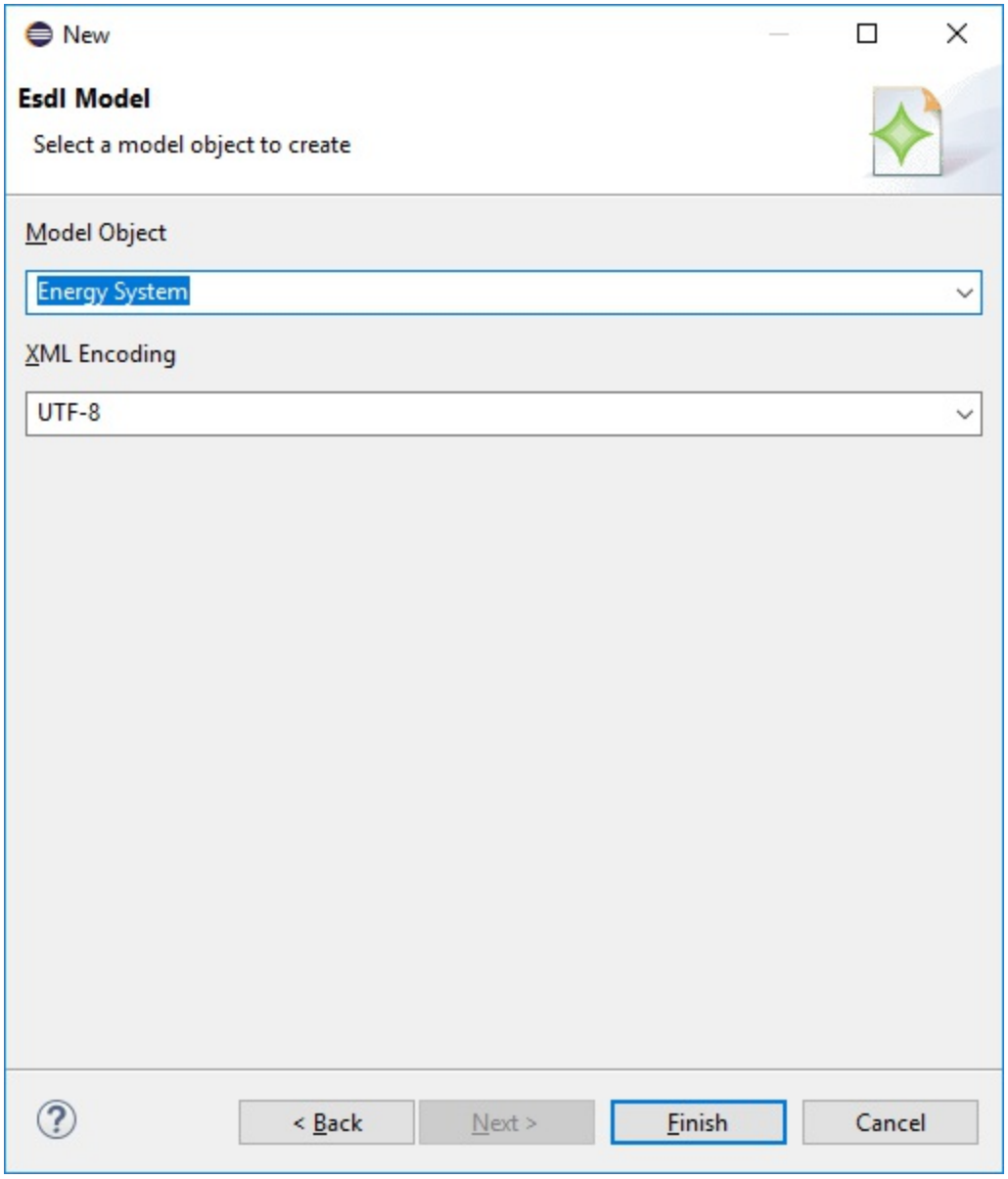

## Step 2. Create a new ESDL model

Select File --> New --> Other and select 'ESDL model' in the dialog box that appears. Presse 'Next'



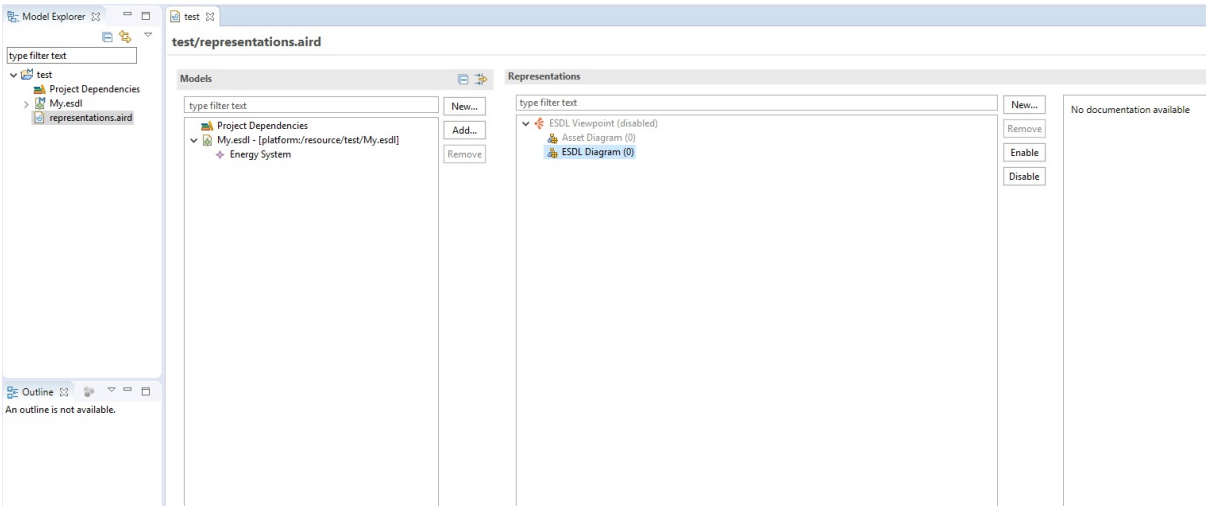

# Step 3. Select 'Energy System'

Select 'Energy System' from the drop-down list under 'Model Object' and click 'Finish'

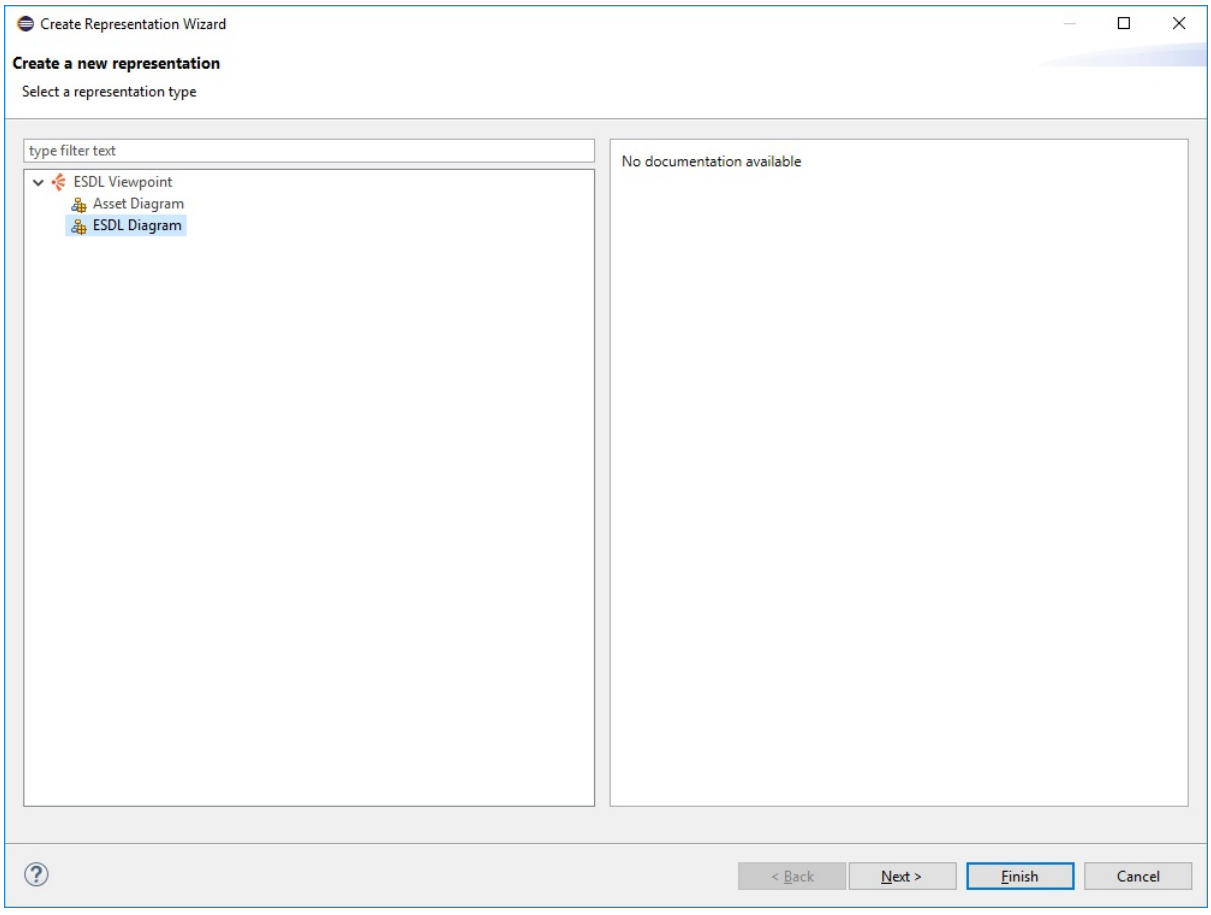



## Step 4. Create a new model representation

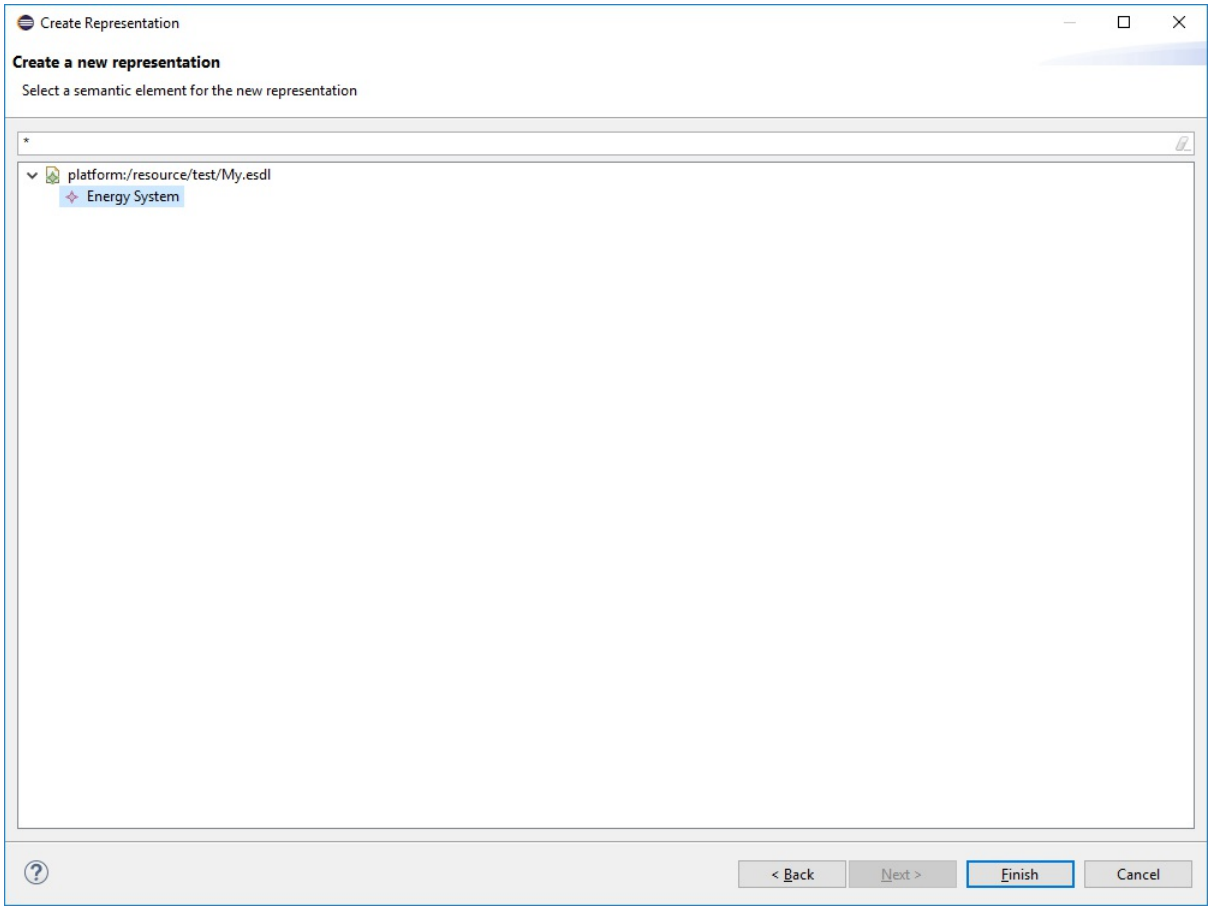
Double click the `representations.aird` file in the project, and click 'New...' under the 'Representations' section.



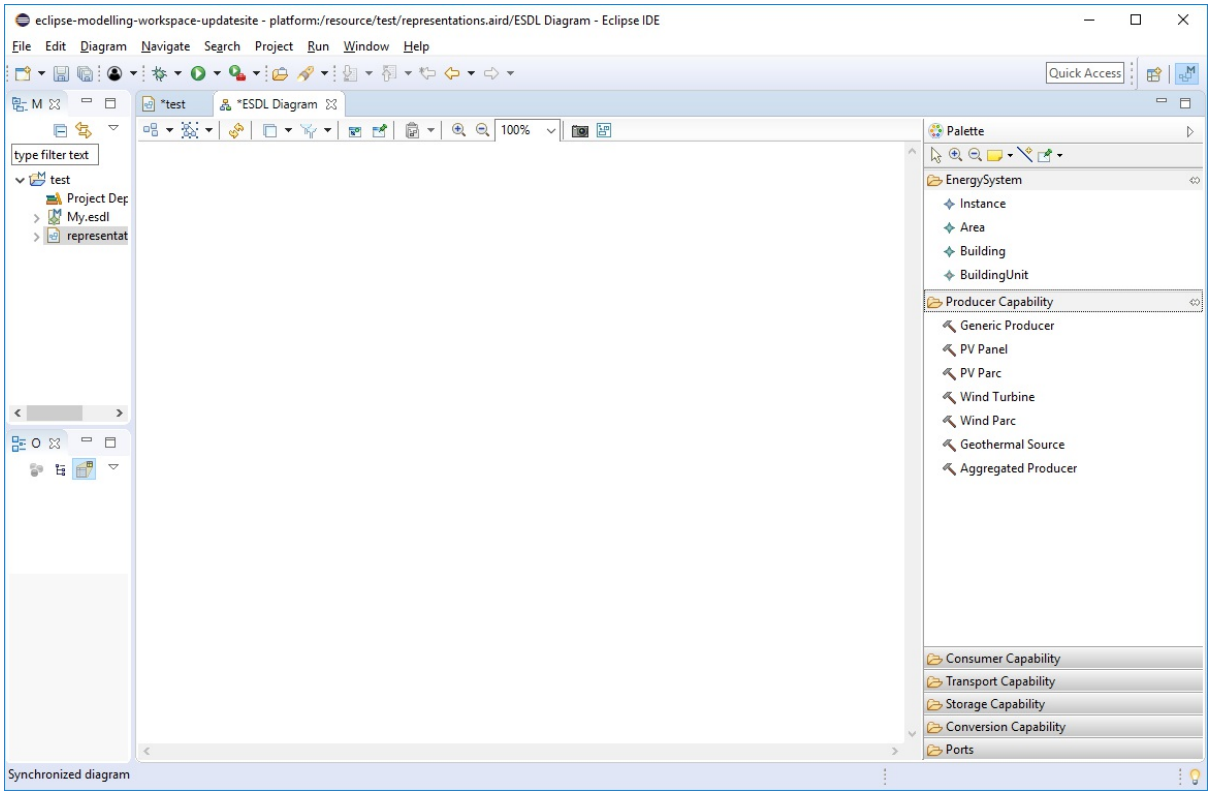

# Step 5. Select 'ESDL Diagram' and click 'Next'



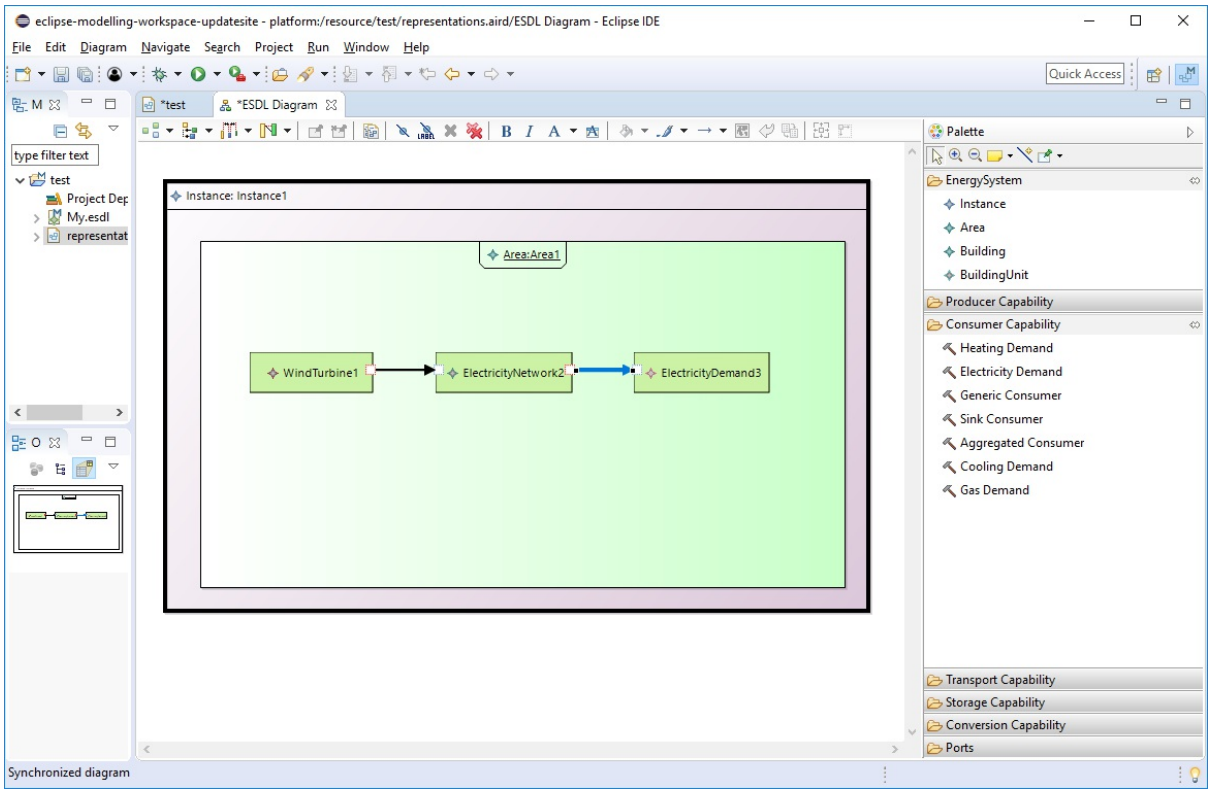

# Step 6. Select the 'Energy System' from your ESDL model and click 'Finish'




# Step 7. You now see the canvas in front of you



# Step 8. Add an Instance, an Area, some Assets and connect them



# Contributing to the ESDL model

To contribute to the ESDL language you have to install Eclipse and the Eclipse Modelling Tools which contain the Eclipse Modelling Framework.

These can be downloaded from the eclipse.org website. Easiest is downloading the modelling package directly with a new Eclipse installation, but alternatively you could also install these modelling tools as plugins in your existing Eclipse installation. Since EMF is still under active development, and ESDL is using new features from these latest releases, it is required to use the latest version of Eclipse.

Currently Photon is the latest stable release of Eclipse. You can download the Eclipse installer from the general download page and select the Eclipse Modelling Tools from the list of options after running the executable.

# Setting up the Eclipse Modelling Tools

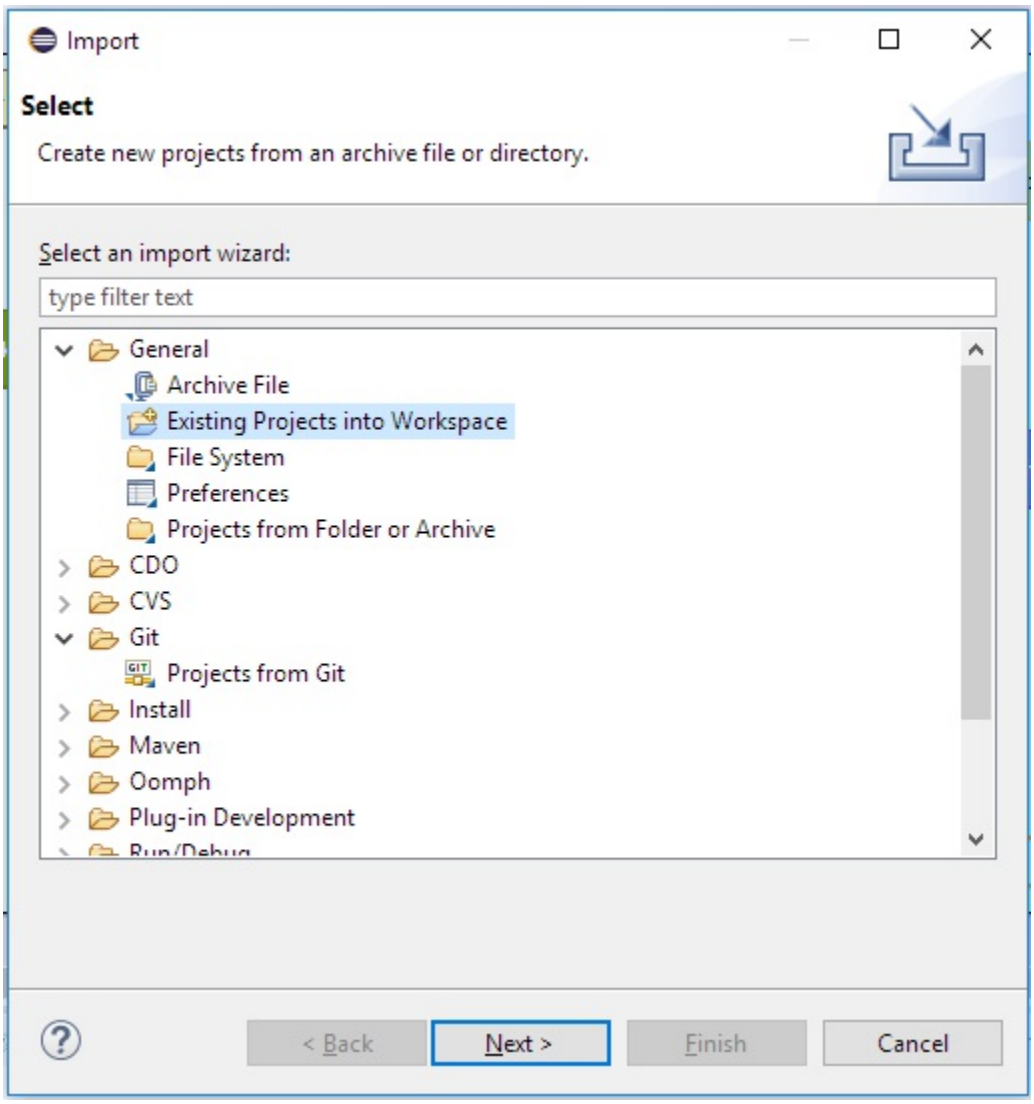
## Download Eclipse Modelling tools

Currently Photon is the latest stable release of Eclipse. You can download the Eclipse installer from the general download page and select the Eclipse Modelling Tools from the list of options after running the executable.

## Setting up Eclipse Modelling Tools

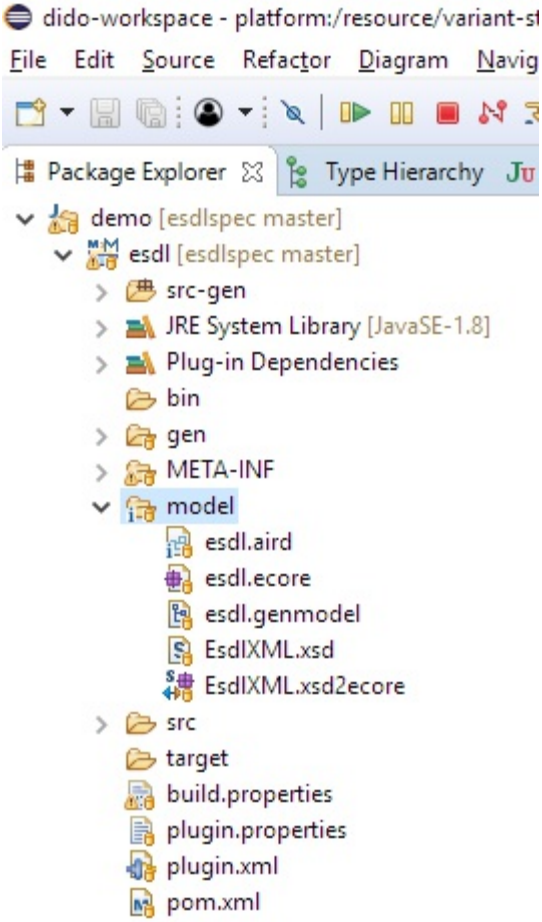
After downloading and installing Eclipse and Eclipse Modelling Tools, you start Eclipse by double clicking on the created icon on the Desktop (if you are using Windows...). It will ask you where you want to place your workspace files. Select a logical location and you end up with a welcome screen.

If you did not clone the repository yet, now is a good moment with your preferred GIT tool or use Eclipse to import the repository directly.



Next step is importing the ESDL git repository by using File -> Import... If you already cloned the repository locally, use *Existing Project into Workspace* and the project will be imported. If you want to directly import the project from Git, use Git->Projects from Git.

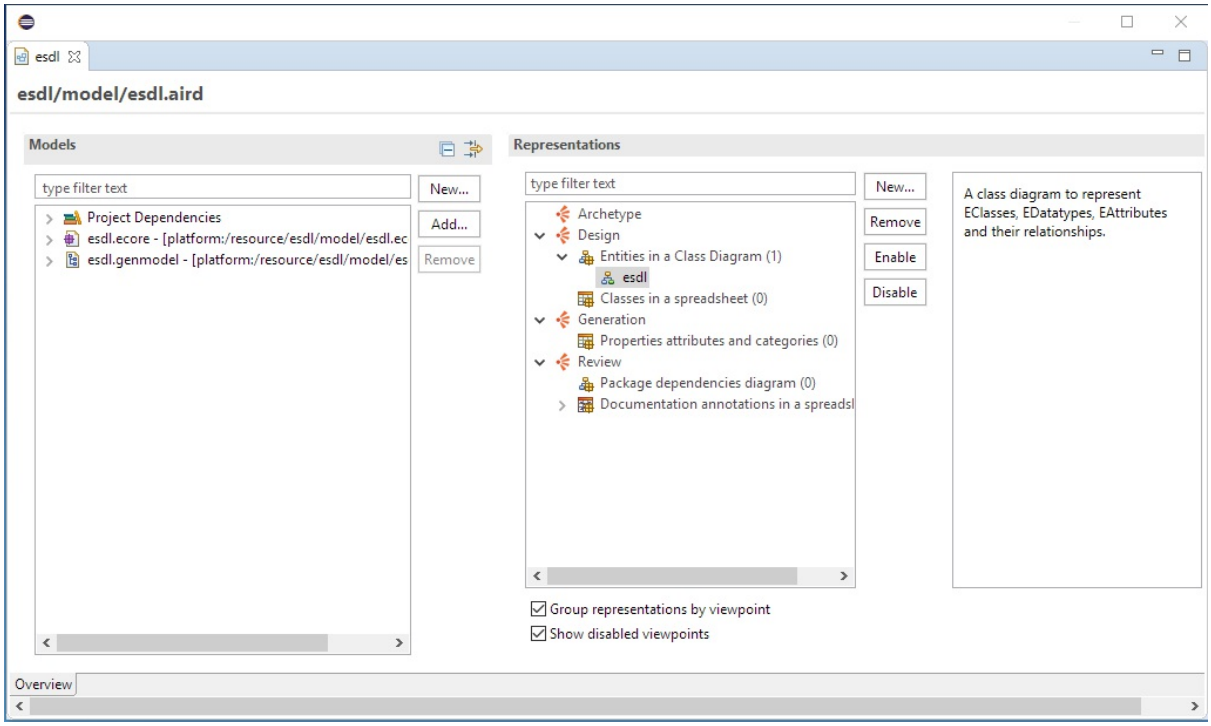
After importing, the *Package explorer* will show the following files and directories:




In the model folder you see several files:

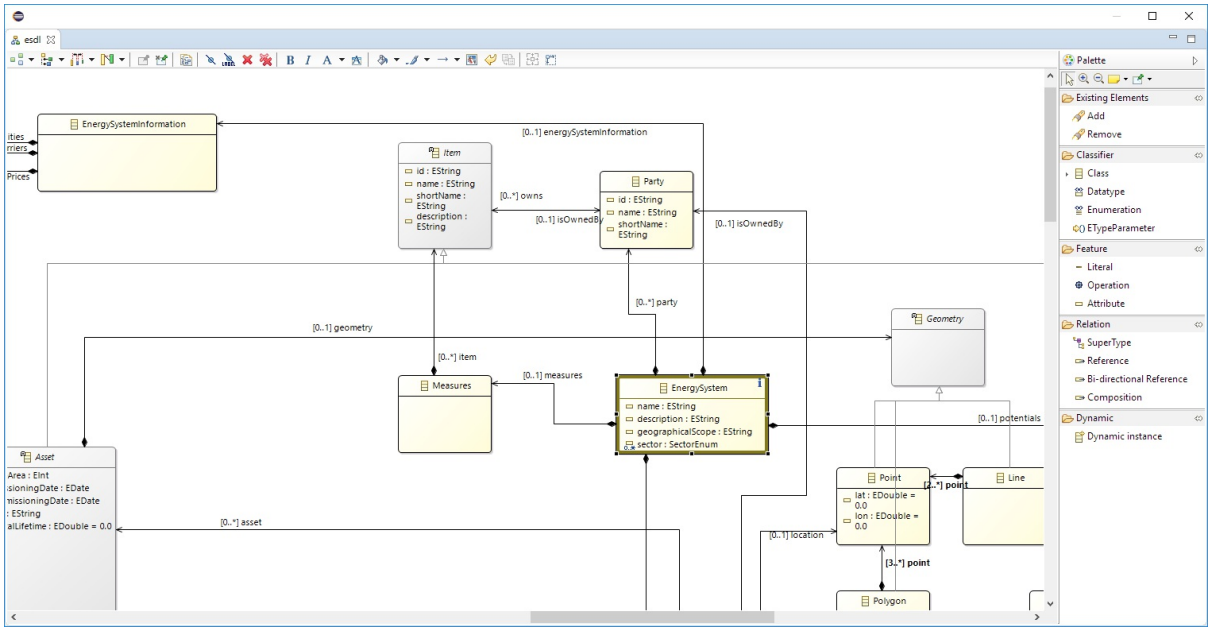
- `esdl.aird` - In this file the graphical representation of the ESDL concepts are documented, such as the (x,y) location of the classes, the colour of the classes, the font, etc. It allows us to visually modify the contents of the ESDL model. If you double click you can open the ESDL model diagram editor and manipulate the model.
- `esdl.ecore` - This is ESDL. The UML-based model describing all ESDL concepts: the vocabulary and grammar of an Energy System. If you double click you can edit the ESDL model using a tree-based structure (less convenient than the diagram editor)
- `esdl.genmodel` - This file describes how to generate Java-files and XSD Schemas from an `ecore`-file. It contains a dozen configuration options.
- `esdlXML.xsd` - this is the XML Schema file for ESDL
- `esdlXML.xsd2ecore` - maps the XSD to the `Ecore` model.

Double clicking on the aird-file opens the following dialog:




The AIRD file shows the model dependencies and representation. The Design->Entities in a Class Diagram is the most interesting. Double click on `esdl` to open this representation.

There is also a Review representation that shows all the documentation in the model. (although this is work in progress).




Now you're setup to edit the Energy System Description Language!

# Generating model, edit and editor code

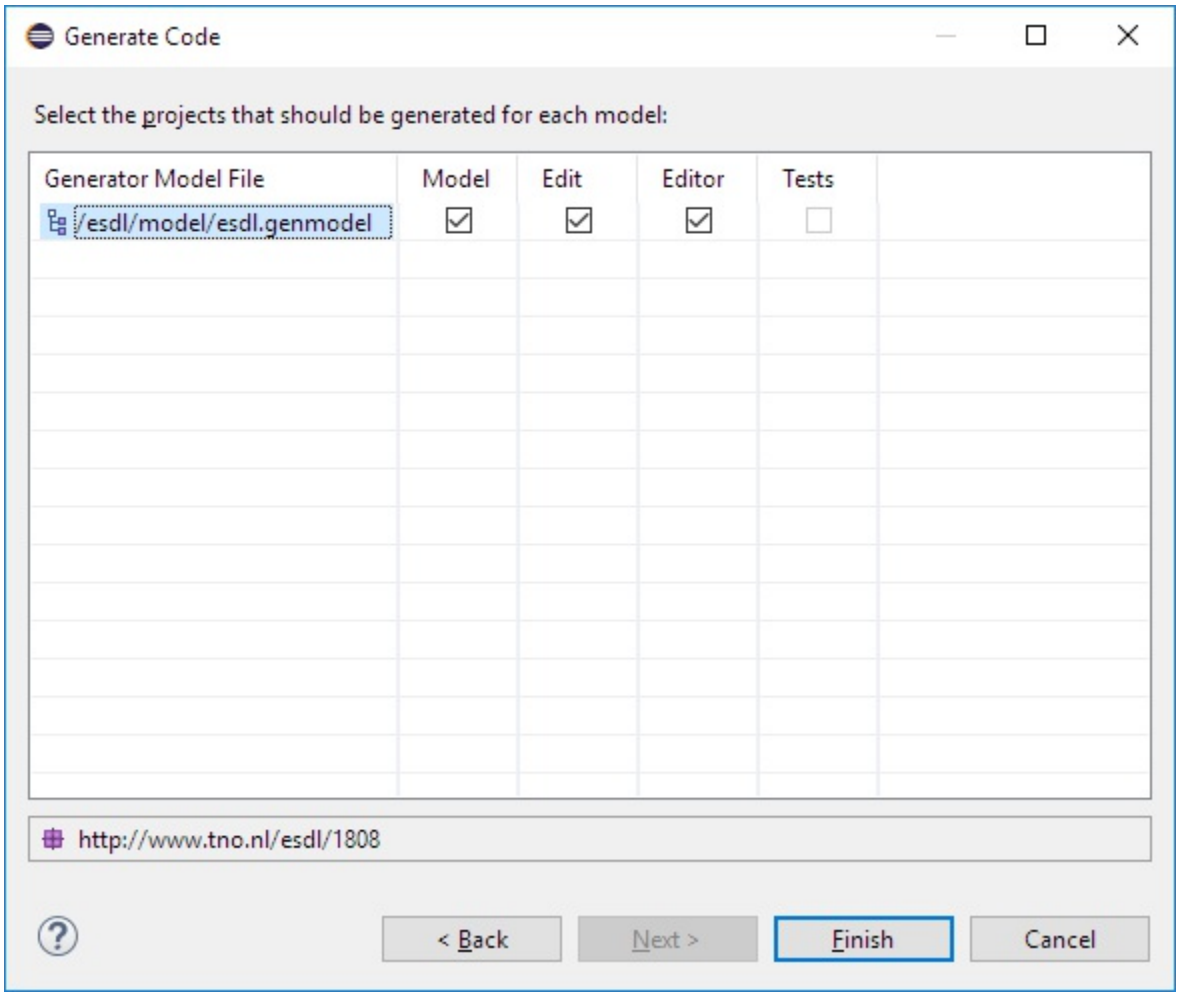
Since Ecore-based models such as the ESDL model contain semantics, source code can be generated from this model. EMF allows you to create three different types of source code:

- ESDL Model code - creates Java-classes out of the model, including the ECore semantics (they inherit from the `EObject-` class)
- ESDL Edit code - creates editor-independent code to manipulate and edit the ESDL model instances.
- ESDL Editor code - creates the Tree-based editor described here to edit ESDL model instances.

## Using the `.genmodel`

Code generation is specified in the `esdl.genmodel` file. In Eclipse you can open this file in its own editor. In the properties there are a lot of options you can configure to change the code generation behaviour.

When right-clicking on the esdl-package you can select Generate... to generate one or more of the above mentioned artefacts, or press Ctrl+Shift+G to pop-up the Generator dialog.




The current `esdl.genmodel` is configured to produce an XML Schema too, that can be used by other non-Java-based software. The next chapters shows how to integrate with e.g. Python.

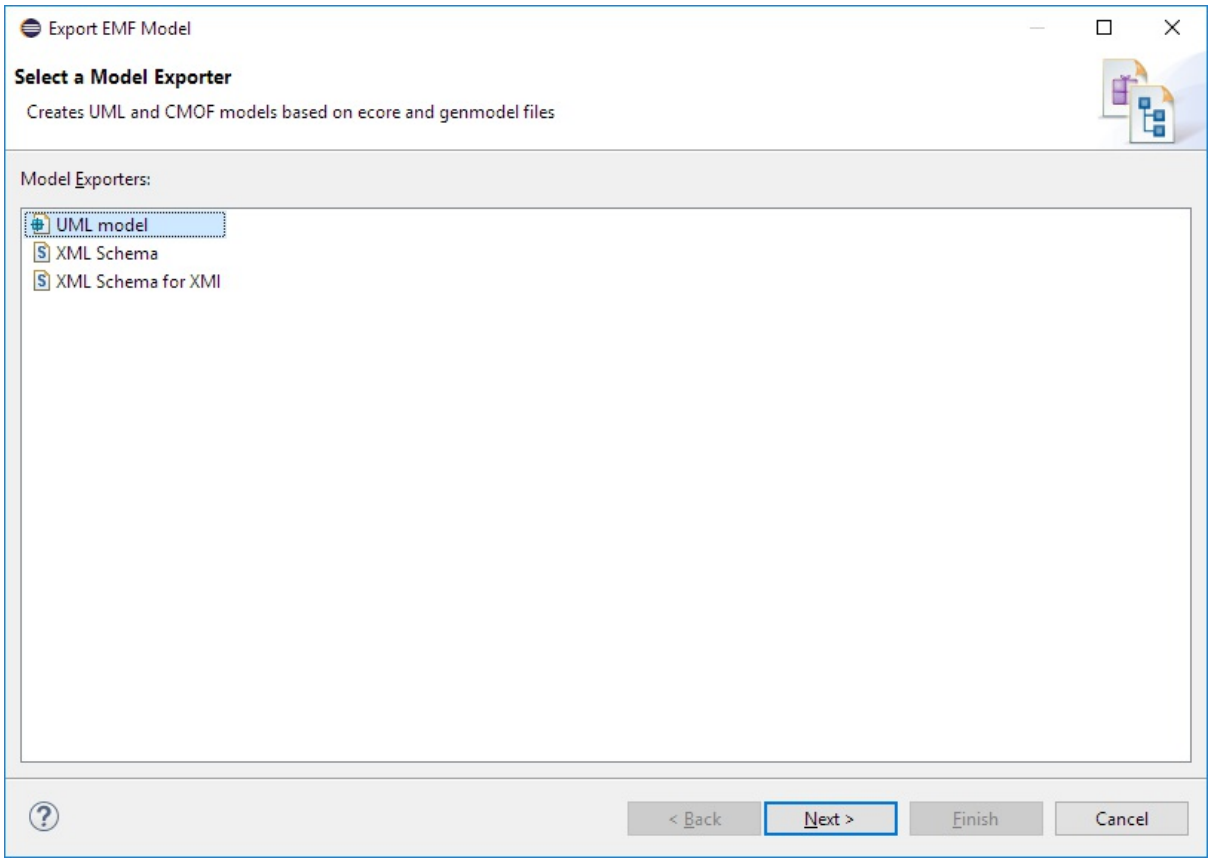
# Integrating ESDL in other applications

ESDL is made to be used in all kinds of applications. Since the ESDL-model is EMF-based, the simplest way is to use the generated model code directly using Java or another JVM-based language.

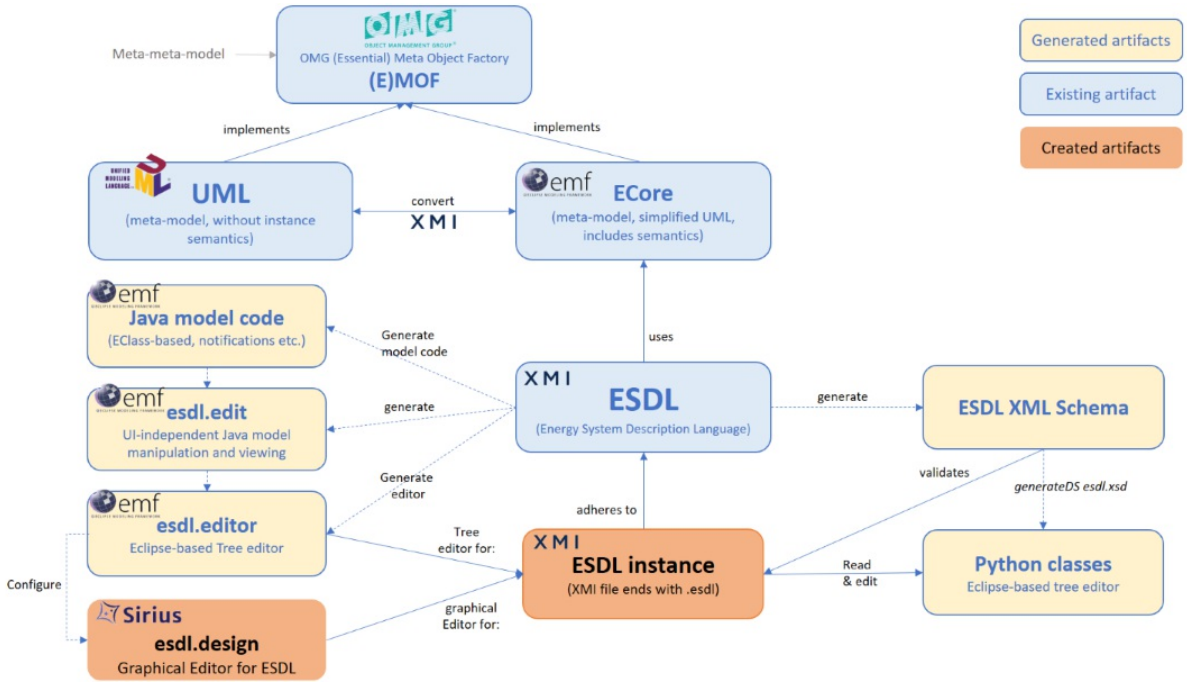
An alternative is using the generated XML-Schema to integrate with non-Java-based programming languages. When code is generated, it automatically creates this XML-schema called `esdlXML.xsd`. Other export options are also available, as EMF stores its models in XMI (XML Metadata Interchange):

- XML Schema for XMI (creating an XMI-based XML schema)
- UML - uses the UML concepts to export to the UML. UML files can subsequently read by other UML modeling tools such as Enterprise Architect or Papyrus UML editor.

You can find these export options by right-clicking on the `esdl.genmodel` file and select "Export model..."




Summarizing the following toolchain is currently available for ESDL (and in use):


Meta-meta-model
OMG (Essential) Meta Object Factory
(E)MOF
implements
implements
UML
(meta-model, without instance semantics)
convert
XMI
ECore
(meta-model, simplified UML, includes semantics)
Java model code
(EClass-based, notifications etc.)
Generate model code
uses
XMI
ESDL
(Energy System Description Language)
generate
ESDL XML Schema
esdl.edit
UI-independent Java model manipulation and viewing
generate
Generate editor
validates
generateDS esdl.xsd
esdl.editor
Eclipse-based Tree editor
adheres to
Tree editor for:
XMI
ESDL instance
(XMI file ends with .esdl)
Read & edit
Python classes
Eclipse-based tree editor
Configure
Sirius
esdl.design
Graphical Editor for ESDL
graphical Editor for:
Generated artifacts
Existing artifact
Created artifacts

# Integration with Java

There are two ways to integrate with Java-based technology:

1. **Use the Eclipse platform** This is currently used for the Edit, Editor and Designer plugins: they are Eclipse plugins that leverage the feature-rich Eclipse platform. Explaining how to do this can be read elsewhere on the internet. Options include using Sirius to create the Designer plugin, using EMF Forms to create attractive UIs to edit model instances, or creating a stand-alone ESDL-based application using Eclipse RCP.
2. **Use the main EMF-jars in your stand-alone Java application**
   If you extract the main EMF-jar files from the EMF plugins or use the Maven repo to download these dependencies into your Maven-project. You require the following EMF-jars:

   ```
   org.eclipse.emf.common_2.14.0.v20xxyyzz.jar
   org.eclipse.emf.ecore_2.14.0.v20xxyyzz.jar
   org.eclipse.emf.ecore.xmi_2.14.0.v20xxyyzz.jar
   ```

The maven repo is currenly a little behind the lastest EMF-versions, but you need the following dependencies in your `pom.xml` :

```
        <dependency>
                <groupId>org.eclipse.emf</groupId>
                <artifactId>org.eclipse.emf.ecore</artifactId>
                <version>2.12.0</version>
        </dependency>
        <dependency>
                <groupId>org.eclipse.emf</groupId>
                <artifactId>org.eclipse.emf.common</artifactId>
                <version>2.12.0</version>
        </dependency>
        <dependency>
                <groupId>org.eclipse.emf</groupId>
                <artifactId>org.eclipse.emf.ecore.xmi</artifactId>
                <version>2.12.0</version>
        </dependency>
```

If these dependencies are added to your application, ESDL can be used in the Java code.

E.g. creating an Energy System with an Area programmatically:

```
EnergySystem energySystem = EsdlFactory.eINSTANCE.createEnergySystem();
energySystem.setName("My First EnergySystem");
energySystem.setId("firstEnergySystem");
Area area = EsdlFactory.eINSTANCE.createArea();
area.setId("Test");
area.setName("Amsterdam municipality")
energySystem.getArea().add(area);
```

The following code allows you to load and save ESDL-model instances:

# Loading of an ESDL-file

```
public static EnergySystem loadESDLModel(String fileName) throws IOException {
```

```
        // Initialize the model
        EsdlPackage.eINSTANCE.eClass();
        XMIResource resource = new XMIResourceImpl(URI.createURI(fileName));
        resource.load(null);
        return (EnergySystem) resource.getContents().get(0);
}
```

If the models grow large with a lot of internal references, add the following lines to speed up the loading of the files:

```
public static EnergySystem loadESDLModel(String fileName) throws IOException {
        // Initialize the model
        EsdlPackage.eINSTANCE.eClass();
        XMIResource resource = new XMIResourceImpl(URI.createURI(fileName));
        // speed up loading of large files by defering ID references lookup.
        resource.getDefaultLoadOptions().put(XMIResource.OPTION_DEFER_IDREF_RESOLUTION, Boolean.TRUE);
        resource.setIntrinsicIDToEObjectMap(new HashMap<String, EObject>());
        // load the resource
        resource.load(null);
        return (EnergySystem) resource.getContents().get(0);
}
```

# Saving an ESDL-file

Saving to an ESDL-file is as follows:

```
public static XMIResource saveESDLModel(EnergySystem energySystem, String fileName) throws IOException {
        XMIResource resource = new XMIResourceImpl(URI.createURI(fileName));
        resource.getContents().add(energySystem);
        HashMap<String, Object> opts = new HashMap<String, Object>();
        // Produce an xsi:schemaLocation in the resource
        opts.put(XMIResource.OPTION_SCHEMA_LOCATION, true);
        resource.save(opts);
        return resource;
    }
```

# Integration with Python

## Introduction

Using the `generateDS` tool with the XSD generated from the ESDL model, you can generate Python code to work with ESDL. The generated code contains the python classes for each ESDL concept and contains functionality to instantiate a new model, add assets to it, read an ESDL description from disk or save one.

## Generate the XSD from the ESDL model

The ESDL GitHub project contains the XML schema definition (XSD) file in the `model` subdirectory.

To generate a new XSD from an updated ESDL model, right click the `esdl.genmodel` file and select the 'Export model...' option from the drop down menu. In the wizard select the 'XML Schema' model exporter and follow the steps in the wizard to create the XSD.

## Create Python classes using GenerateDS

The GenerateDS project provides us with a tool to generate Python data structures and an XML parser from an XML schema definition (XSD).

You can find the GenerateDS project here.

Install generateDS with `pip install generateDS` to add it to your python libraries

Using the following command, you can create the python code from the XSD file:

```
generateDS -o esdl.py .\esdl\EsdlXML.xsd
```

## Using the generated Python code

Use the following code to load an ESDL-file:

```
def load_energy_system(inpath):
    with open(inpath, 'r') as infile:
        # parse esdl file
        es = esdl.parse(infile, silence=False)
        return es

es = load_energy_system('Amsterdam.esdl')
# get all assets of an area within the energy system (use 1st instance only)
assets = energy_system.get_instance()[0].get_area().get_asset()
print(assets)
```

# Examples

The variety of possible applications of ESDL is enormous. The language allows both very detailed descriptions (individual house level) as very high level descriptions (national level). This sections will show some examples of applications of ESDL to give an expression of the possibilities.

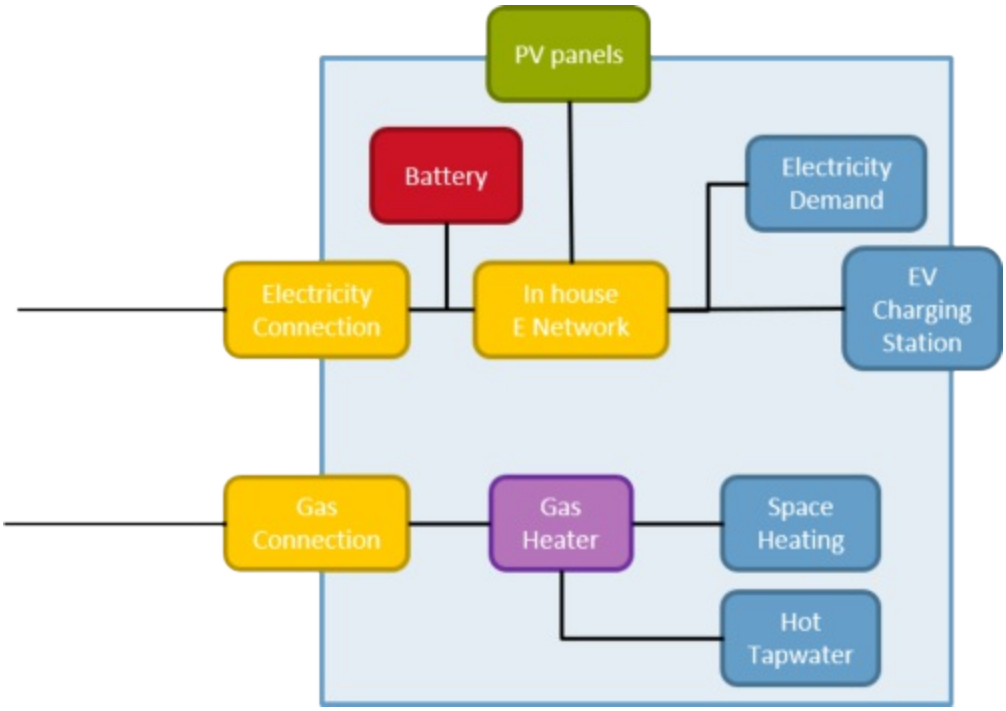
Examples are:

- Describing a house
- Describing a municipality

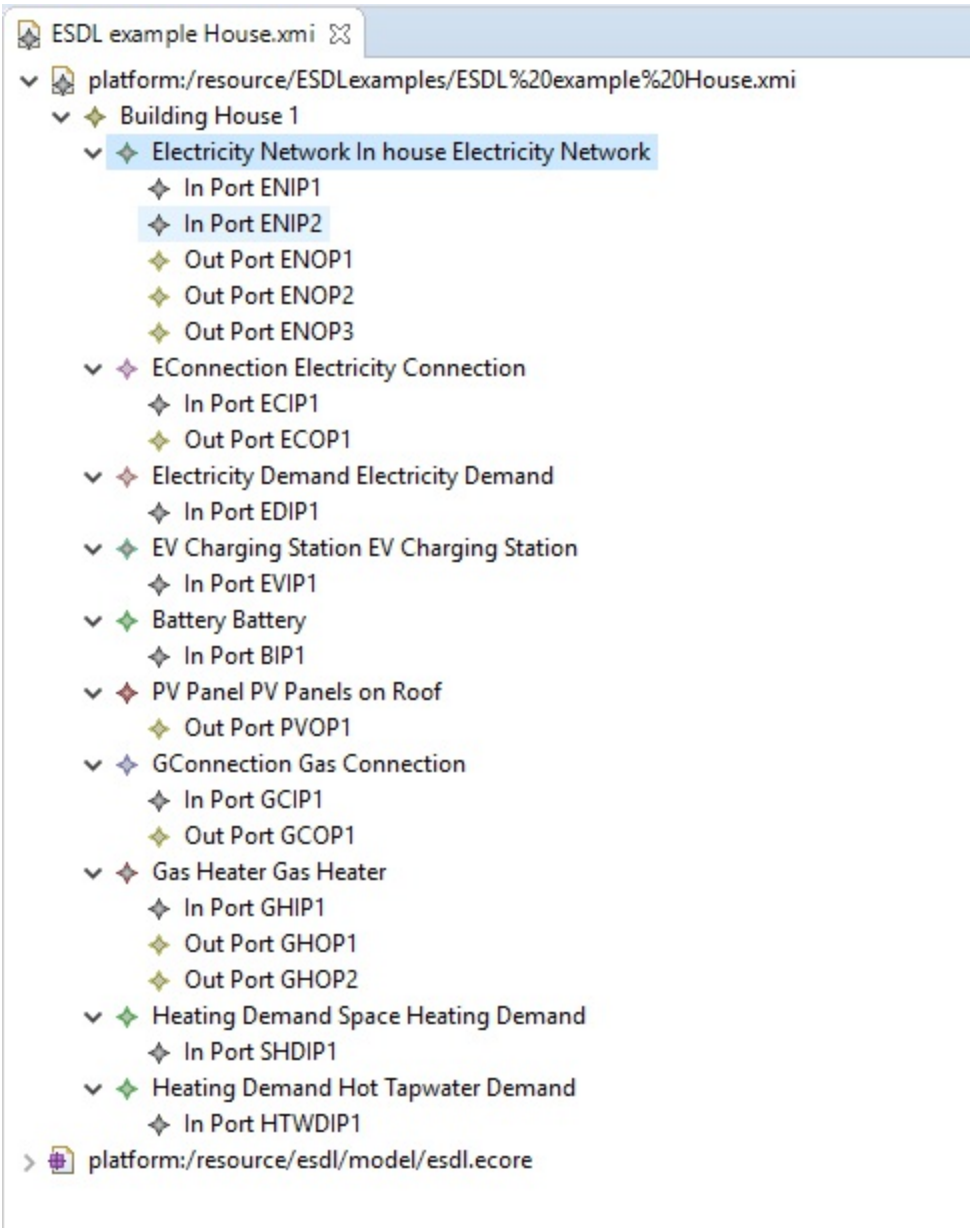
More examples will be added in the coming months.

# Describing a house

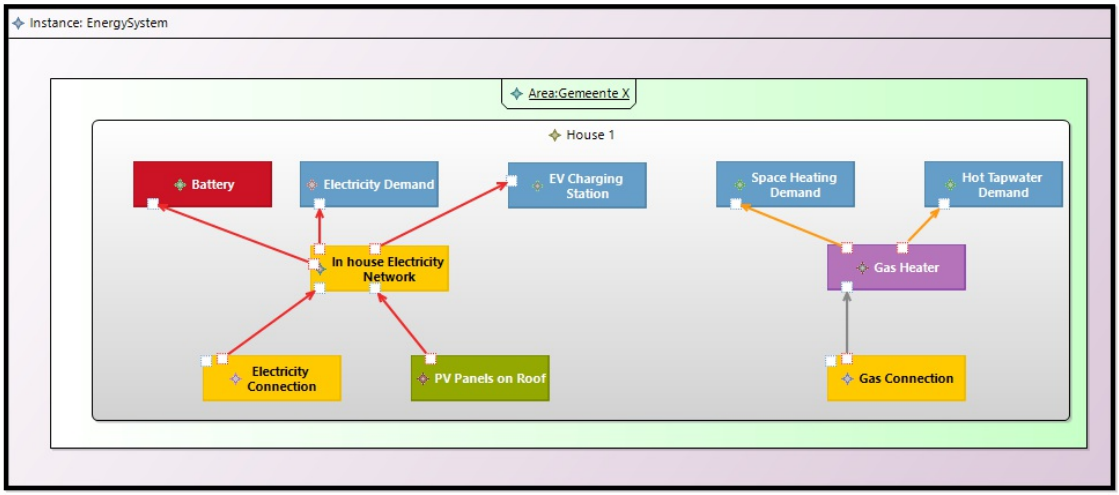
An example house as shown in the picture below can easily be described in ESDL.




In the eclipse tree editor the house would look like this:




In the eclipse graphical desginer the house (with an EnergySystem, Instance and Area around it) would look like this:




The ESDL would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<esdl:Building
    xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:esdl="http://www.tno.nl/esdl/1807a"
    xsi:schemaLocation="http://www.tno.nl/esdl/1807a ../esdl/model/esdl.ecore"
    id="H1"
    name="House 1"
    commissioningDate="2010-07-15T01:00:00.000+0200"
    energyLabel="LABEL_D"
    slantedRoofArea="60.0">
  <asset xsi:type="esdl:ElectricityNetwork"
      id="EN1"
      name="In house Electricity Network"
      voltage="230.0">
    <port xsi:type="esdl:InPort"
        id="ENIP1"
        connectedTo="ECOP1"/>
    <port xsi:type="esdl:InPort"
        id="ENIP2"
        connectedTo="PVOP1"/>
    <port xsi:type="esdl:OutPort"
        id="ENOP1"
        connectedTo="BIP1"/>
    <port xsi:type="esdl:OutPort"
        id="ENOP2"
        connectedTo="EVIP1"/>
    <port xsi:type="esdl:OutPort"
        id="ENOP3"
        connectedTo="EDIP1"/>
  </asset>
  <asset xsi:type="esdl:EConnection"
      id="EC1"
      name="Electricity Connection">
    <port xsi:type="esdl:InPort"
        id="ECIP1"/>
    <port xsi:type="esdl:OutPort"
        id="ECOP1"
        connectedTo="ENIP1"/>
  </asset>
  <asset xsi:type="esdl:ElectricityDemand"
      id="ED1"
      name="Electricity Demand">
```

```
  <port xsi:type="esdl:InPort"
      id="EDIP1"
      connectedTo="ENOP3"/>
</asset>
<asset xsi:type="esdl:EVChargingStation"
    id="EV1"
    name="EV Charging Station">
  <port xsi:type="esdl:InPort"
      id="EVIP1"
      connectedTo="ENOP2"/>
</asset>
<asset xsi:type="esdl:Battery"
    id="B1"
    name="Battery">
  <port xsi:type="esdl:InPort"
      id="BIP1"
      connectedTo="ENOP1"/>
</asset>
<asset xsi:type="esdl:PVPanel"
    id="PV1"
    name="PV Panels on Roof"
    surfaceArea="16"
    power="3000.0">
  <port xsi:type="esdl:OutPort"
      id="PVOP1"
      connectedTo="ENIP2"/>
</asset>
<asset xsi:type="esdl:GConnection"
    id="GC1"
    name="Gas Connection">
  <port xsi:type="esdl:InPort"
      commodity="GAS"
      id="GCIP1"/>
  <port xsi:type="esdl:OutPort"
      commodity="GAS"
      id="GCOP1"
      connectedTo="GHIP1"/>
</asset>
<asset xsi:type="esdl:GasHeater"
    id="GH1"
    name="Gas Heater"
    efficiency="90.0">
  <port xsi:type="esdl:InPort"
      commodity="GAS"
      id="GHIP1"
      connectedTo="GCOP1"/>
  <port xsi:type="esdl:OutPort"
      commodity="HEAT"
      id="GHOP1"
      connectedTo="SHDIP1"/>
  <port xsi:type="esdl:OutPort"
      commodity="HEAT"
      id="GHOP2"
      connectedTo="HTWDIP1"/>
</asset>
<asset xsi:type="esdl:HeatingDemand"
    id="SHD1"
    name="Space Heating Demand">
  <port xsi:type="esdl:InPort"
      commodity="HEAT"
      id="SHDIP1"
      connectedTo="GHOP1"/>
</asset>
<asset xsi:type="esdl:HeatingDemand"
    id="HTW1"
    name="Hot Tapwater Demand">
```

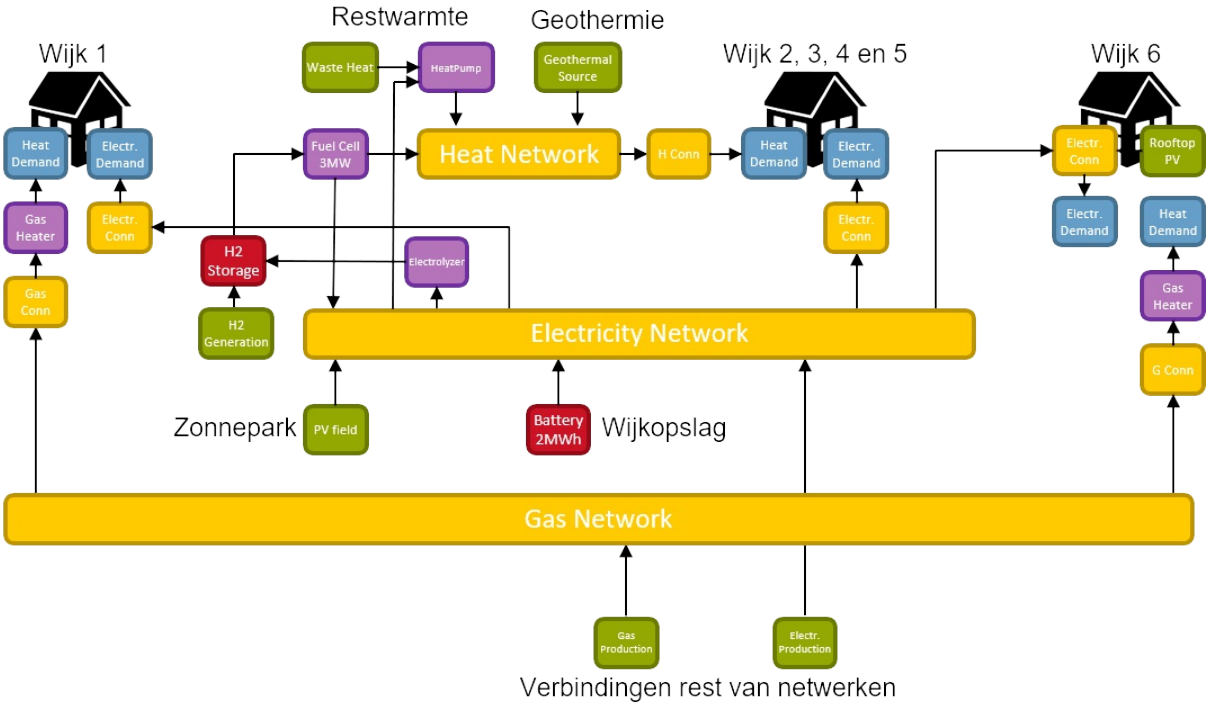
```
    <port xsi:type="esdl:InPort"
        commodity="HEAT"
        id="HTWDIP1"
        connectedTo="GHOP2"/>
  </asset>
</esdl:Building>
```

# Describing a municipality

An example municipality as shown in the picture below can easily be described in ESDL.

A similar municipality example is published on GitHub at the following URL:

https://github.com/EnergyTransition/ESDL-municipality-example



# Contact

See for contact information this site.

# ESDL Release Notes

## Release v1808 (link)

- Merged Commodities and EnergyCarriers lists into Carriers list
- Removed commodities enumeration from ports
- Added mobility concepts to ESDL (Mobility demand per vehicle type and fuel type, fuel efficiency per vehicle type, number of vehicles per area)
- Improved the Geometry classes (added MultiPolygon)
- Added assets (ResidualHeatSource, MobilityDemand, SolarCollector, FermentationPlant)
- Added residual heat source potential
- Added Parties class to model ownership
- Added DataSources class to list sources of used data in an ESDL model
- Added Profiles class and ReferenceProfile class to enable reuse of profiles
- Measures is an Asset collection instead of an Item collection (to ease development of ESDL designer)

## Release v1807a (link)

- Generated proper XSD, the v1807 release still contained the v1806 XSD

## Release v1807 (link)

- Added 'UNDEFINED' and 'ENERGY' to CommodityEnum
- Renamed some assets:
  - HeatPipe --> Pipe
  - HeatDemand --> HeatingDemand
  - CoolDemand --> CoolingDemand

## Release v1806 (link)

- Initial public release