1 Test Case 01-001: Fully qualified element name (FQEN)

1.1 Description

Parsing a `*.sysml` file will generate a fully qualified element name (FQEN) for each element, starting with `Root`.

1.2 Scope

The scope of this Test Case is uSysML v0.01, and the applicable keywords are: `package`, `part def`, and `part`.

1.3 SysML v2 textual notation

```
package PackageVehicles {

    part def Vehicle;
    part def Wheel;

    part vehicle:Vehicle {
        part w:Wheel;
    }
}
```

1.4 uSysML output

```
Root.PackageVehicles [Package]
 Root.PackageVehicles.Vehicle [PartDef]
 Root.PackageVehicles.Wheel [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.w [PartUsage]
      typed by=Root.PackageVehicles.Wheel
```

1.5 Discussion

Within a sysml file all fully qualified element names (FQENs) must be uniqe. The following example should generate an error, because both `part def Wheel` and `part Wheel` have the same FQEN `PackageVehicles.Wheel`:

```
package PackageVehicles {

    part def Vehicle;
    part def Wheel;
    part Wheel;

    part vehicle:Vehicle {
        part w:Wheel;
    }
}
```

A potential issue arises when multiple `*.sysml` files are processed. Let's assume two files, `Vehicles1.sysml` and `Vehicles2.sysml` with the following content:

–file Vehicles1.sysml–

```
part def Vehicle;
part def Wheel;
```

–file Vehicles1.sysml–

```
part def Vehicle;
```

Let's assume a SysMLv2 processor `sysmlv2` which can take one or more `*.sysml` files on the input. Then processing the first file will result in:

```
$ sysmlv2 Vehicles1.sysml
Root.Vehicle [PartDef]
Root.Wheel [PartDef]
```

And processing the second file will result in:

```
$ sysmlv2 Vehicles2.sysml
Root.Vehicle [PartDef]
```

When the two files are processed together:

```
$ sysmlv2 Vehicles1.sysml Vehicles2.sysml
```

the FQENs for the element `Root.Vehicle` will create a name clash.

1.6 Notes

NOTE 1: This test case assumes that each `*.sysml` file defines the namespace for the elements it contains. The top level namespace bounded by the file is assigned to the element `Root`.

NOTE 2: The Pilot implementation pocesses the example:

```
package PackageVehicles {

    part def Vehicle;
    part def Wheel;
    part Wheel;

    part vehicle:Vehicle {
        part w:Wheel;
    }
}
```

with no errors, but issues the following warnings:

```
WARNING:Duplicate owned member name (1.sysml line : 4 column : 14)
WARNING:Duplicate owned member name (1.sysml line : 5 column : 10)
```

2 Test Case 01-002: *PartDefinition* element

2.1 Description

Explores characterists of *PartDefinition* element.

2.2 Scope

The scope of this Test Case is uSysML v0.01, and the applicable keywords are: `package`, `part def`, and `part`.

2.3 SysML v2 textual notation

```
package PackageVehicles {

    part def Vehicle;

    part def Wheel {
        part def Lugbolt;
    }

    part vehicle:Vehicle {
        part w:Wheel;
    }
}
```

2.4 uSysML output

```
Root.PackageVehicles [Package]
 Root.PackageVehicles.Vehicle [PartDef]
 Root.PackageVehicles.Wheel [PartDef]
  Root.PackageVehicles.Wheel.Lugbolt [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.w [PartUsage]
      typed by=Root.PackageVehicles.Wheel
```

2.5 Discussion

The keyword `part def` declares a part definition element. A part defintion element is called *PartDefinition* in Sysml v2 abstract syntax diagrams, and is denoted '[PartDef]' in the uSysML output. Speaking in general, the purpose of SysML v2 definition elements is to type appropriate usage elements. The usage element that can be typed by *PartDefinition* is *PartUsage*, declared with the keyword `part`. *PartUsage* is a usage element that represent usage of a part definition. Also see Test Case 01-003.

In the textual notation example above, the following elements are *PartDefinition*: * `part def Vehicle;` – 'Vehicle' is *PartDefinition* without the body specified * `part def Wheel { part def Lugbolt; }` – 'Wheel' is *PartDefinition* with a body which defines its namespace, and declares another *PartDefinition* 'Lugbolt'.

The nested elements within `part def` are referred to as its features. Thus in the above example `Wheel` is *PartDefinition* with one feature, `Lugbolt` (which itself is also *PartDefinition*).

2.6 Notes

In SysML v2 the base type of every *PartDefinition* is `Parts::Part` from the Systems Library. Thie means that every *PartDefintion* is a subclass of Systems Library `Parts::Part`, directly or indirectly. Systems Library `Parts::Part` is itself a *PartDefinition*, as given below:

```
package Parts {
    private import Objects::Object;
    private import Objects::objects;
    private import Items::Item;
    private import Items::items;
    private import Ports::Port;
    private import Ports::ports;
    private import Actions::Action;
    private import States::StateAction;

    abstract part def Part :> Item {
        ref self: Part :>> Item::self;
        part start: Part :>> Item::start;
        part done: Part :>> Item::done;
        abstract port portsOnPart: Port[0..*] :> ports;
        abstract ref action performedActions: Action[0..*] :> enactedPerformances;
        abstract ref state exhibitedStates: StateAction[0..*] :> performedActions;
    }
    abstract part parts: Part[0..*] nonunique :> items;
}
```

We note that `Parts::Part` subsets `Items::Item` from the Systems Library.

3 Test Case 01-003: *PartUsage* element

3.1 Description

Explores characterists of *PartUsage* element.

3.2 Scope

The scope of this Test Case is uSysML v0.01, and the applicable keywords are: `package`, `part def`, and `part`.

3.3 SysML v2 textual notation

```
package PackageVehicles {

    part def Vehicle;

    part def Wheel {
        part def Lugbolt;
    }

    part vehicle:Vehicle {
        part w:Wheel;
        part q;
    }
}
```

3.4 uSysML output

```
Root.PackageVehicles [Package]
 Root.PackageVehicles.Vehicle [PartDef]
 Root.PackageVehicles.Wheel [PartDef]
  Root.PackageVehicles.Wheel.Lugbolt [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.w [PartUsage]
      typed by=Root.PackageVehicles.Wheel
  Root.PackageVehicles.vehicle.q [PartUsage]
      typed by=None
```

3.5 Discussion

The keyword `part` declares a part usage element. A part usage element is called *PartUsage* in the Sysml v2 abstract syntax diagrams, and is denoted as '[PartUsage]' in the uSysML output.

The base Element for *PartUsage* is `Parts::parts` from the Systems Library. *PartUsage* is always typed by *PartDefinition*, either explicitly or implicitly. If *PartUsage* is not typed explicitly its type is determined by its base element. The base element for *PartUsage* is `Parts::parts` from the Systems Library, which itself is typed by `Parts::Part` (the base definition for *PartDefinition*).

In this test case example, the following holds: * `vehicle` is *PartUsage* element that is typed explicitly by *PartDefinition* `Vehicle` * `w` is *PartUsage* element that is typed explicitly by *PartDefinition* `Wheel` * `q` is *PartUsage* not typed explicitly (and thus it is implicitly typed with `Parts::parts`)

3.6 Notes

Note 1. There are other possible base types for *PartUsage*: * `RequirementCheck::stakeholders` from the Requirements library model * `RequirementCheck::actors` from the Requirements library model * `Case::actors` from the Cases library model * `Item::subparts` from the Items library model

Note that for each of the above the base Element is typed by *PartDefinition*, and therefore *PartUsage* is always typed by *PartDefinition*.

These base types are outside of the scope of uSysML v0.01 and hence not discussed as a part of this Test Case.

4 Test Case 01-004: *PartUsage* element syntactical forms

4.1 Description

Explores syntactical forms of *PartUsage* element.

4.2 Scope

The scope of this Test Case is uSysML v0.01, and the applicable keywords are: `package`, `part def`, and `part`.

4.3 SysML v2 textual notation

```
package PackageVehicles {

    part def Vehicle;
    part def Wheel;
    part def WheelAxle;

    part test_vehicle:Vehicle;

    part vehicle:Vehicle {
        part a:WheelAxle[2];
        part w:Wheel[4] {
            part def LugBolt;
        }
    }
}

package SupportComponents {

    part parking_space;
    part vehicle_shed[4];

    part repair_shop[2] {
        part def VehicleLift;
    }
}
```

4.4 uSysML output

```
Root.PackageVehicles [Package]
 Root.PackageVehicles.Vehicle [PartDef]
 Root.PackageVehicles.Wheel [PartDef]
 Root.PackageVehicles.WheelAxle [PartDef]
 Root.PackageVehicles.test_vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.a [PartUsage]
      multiplicity=2
      typed by=Root.PackageVehicles.WheelAxle
  Root.PackageVehicles.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles.Wheel
   Root.PackageVehicles.vehicle.w.LugBolt [PartDef]
Root.SupportComponents [Package]
 Root.SupportComponents.parking_space [PartUsage]
    typed by=None
 Root.SupportComponents.vehicle_shed [PartUsage]
    multiplicity=4
    typed by=None
 Root.SupportComponents.repair_shop [PartUsage]
    multiplicity=2
    typed by=None
  Root.SupportComponents.repair_shop.VehicleLift [PartDef]
```

4.5 Discussion

The following syntactical forms of *PartUsage* element are illustrated in the above example:

1. `parking_space` — *PartUsage* without multiplicity, without body, and not explicitely typed
2. `vehicle_shed[4];` — *PartUsage* with multiplicity, without body, and not explicitely typed
3. `repair_shop[2] { ... }` — *PartUsage* with multiplicity, with body, and not explicitely typed
4. `test_vehicle:Vehicle;` — *PartUsage* without multiplicity, without body, and typed explicitly
5. `vehicle:Vehicle { ... }` — *PartUsage* without multiplicity, with body, and typed explicitly
6. `a:WheelAxle[2];` — *PartUsage* with multiplicity, without body, and typed explicitly
7. `w:Wheel[4] { ... }` — *PartUsage* with multiplicity, with body, and typed explicitly

4.6 Notes

None.

5 Test Case 01-005: Namespace search rules

5.1 Description

Illustrates namespace search rules.

5.2 Scope

The scope of this Test Case is uSysML v0.01, the applicable keywords are: `package`, `part def`, and `part`.

5.3 SysML v2 textual notation

```
package PackageVehicles1 {

    part def Vehicle;
    part def Wheel {

    }

    part vehicle:Vehicle {
        part w:Wheel[4];
    }
}

package PackageVehicles2 {

    part def Vehicle;
    part def Wheel;

    part vehicle:Vehicle {
        part def Wheel;
        part w:Wheel[4] {
            part lb:Lugbolt;
        }
    }
}

part def Lugbolt;
```

5.4 uSysML output

```
Root.PackageVehicles1 [Package]
 Root.PackageVehicles1.Vehicle [PartDef]
 Root.PackageVehicles1.Wheel [PartDef]
 Root.PackageVehicles1.vehicle [PartUsage]
```

```
    typed by=Root.PackageVehicles1.Vehicle
  Root.PackageVehicles1.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles1.Wheel
Root.PackageVehicles2 [Package]
 Root.PackageVehicles2.Vehicle [PartDef]
 Root.PackageVehicles2.Wheel [PartDef]
 Root.PackageVehicles2.vehicle [PartUsage]
    typed by=Root.PackageVehicles2.Vehicle
  Root.PackageVehicles2.vehicle.Wheel [PartDef]
  Root.PackageVehicles2.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles2.vehicle.Wheel
```

5.5 Discussion

The namespace search starts from its own namespace, then proceeds into the parent namespaces in the order, until the Root namespace is reached. A name may defined in the Root namespace outside of any package.

In package `PackageVehicles2`, *PartDefintion* `Wheel` exists both within the package `PackageVehicles2` and *PartUsage* `vehicle` namespaces (and furthermore, *PartUsage* `vehicle` namespace is nested within the package `PackageVehicles2` namespace). In this case, *PartUsage* `w` is typed by *PartDefinition* `Wheel` that is within the *PartUsage* `vehicle` namespace.

Using FQENs, the following statements are true:

- `Root.PackageVehicles1.vehicle.w` is *PartUsage* typed by *PartDefintion* `Root.PackageVehicles1.Wheel`
- `Root.PackageVehicles2.vehicle.w` is *PartUsage* typed by *PartDefintion* `Root.PackageVehicles2.vehicle.Wheel`
- `PackageVehicles2.Wheel` is *PartDefintion* that isn't used
- `Root.PackageVehicles2.vehicle.w.lb` is *PartUsage* that is typed by `Root.Lugbolt`

6 Test Case 02-001: A single line note ("//"-type comment)

6.1 Description

Shows the use of //-type comment.

6.2 Scope

The scope of this Test Case is uSysML v0.02, the applicable keywords are: `package`, `part def`, and `part`.

6.3 SysML v2 textual notation

```
package PackageVehicles {

    // an indented comment taking the entire line
    part def Vehicle;// an appended comment without white space
    part def Wheel;           // an appended comment with white space

    part vehicle:Vehicle {
        part w:Wheel[4];
    }
}
// a comment taking the entire line
```

6.4 uSysML output

```
Root.PackageVehicles [Package]
 Root.PackageVehicles.Vehicle [PartDef]
 Root.PackageVehicles.Wheel [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
```

```
  Root.PackageVehicles.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles.Wheel
```

6.5 Discussion

A comment embedded within the SysML v2 textual notation is used to annotate the textual notation. There are multiple types of comments in SysML v2 (see KerML 7.1.2.3), and specifically KerML distingushes *notes* and *comments*, where

- *notes* are used to annotate the SysML textual notation and are discarded during parsing
- *comments* are parsed and into to Comment elements and are stored as part of the model

This test case addresses only *note*, and furthermore of the two kind of notes (*singe line note* and *multiline note*) addressed only a single-line note.

6.6 Notes

Note 1. As per KerML 7.1.2.3:

> "A single-line note includes all the text from the initial characters '//' up to the next line terminator or the end of the input text (whichever comes first)."

7 Test Case 02-002: Unrestricted names (basic form)

7.1 Description

Shows the use of unrestricted element names, and specifically how names can contain spaces when enclosed by single quotes.

7.2 Scope

The scope of this Test Case is uSysML v0.02, and the applicable keywords are: `package`, `part def`, and `part`.

7.3 SysML v2 textual notation

```
package 'My PackageVehicles' {

    part def 'Vehicle';
    part def Wheel;

    part 'vehicle':Vehicle {
        part w:Wheel[4];
    }
}
```

7.4 uSysML output

```
Root.'My PackageVehicles' [Package]
 Root.'My PackageVehicles'.Vehicle [PartDef]
 Root.'My PackageVehicles'.Wheel [PartDef]
 Root.'My PackageVehicles'.vehicle [PartUsage]
    typed by=Root.'My PackageVehicles'.Vehicle
  Root.'My PackageVehicles'.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.'My PackageVehicles'.Wheel
```

7.5 Discussion

The spiral 1 test cases use basic names. This test case shows use of unrestricted names, but only in the basic form without the use of escape sequences.

As per KerML p20, there are two kinds of names:

1. A basic name is one that can be lexically distinguish in itself from other kinds of tokens. The initial character of a basic name must be one of a lowercase letter, an uppercase letter or an underscore. The remaining characters of a basic name are allowed to be any character allowed as an initial character plus any digit. However, a reserved keyword may not be used as a name, even though it has the form of a basic name (see 7.1.2.7), including the Boolean literals true and false.
2. An unrestricted name provides a way to represent a name that contains any character. It is represented as a non-empty sequence of characters surrounded by single quotes. The characters within the single quotes may not include non-printable characters (including backspace, tab and newline). However, these characters may be included as part of the name itself through use of an escape sequence. In addition, the single quote character or the backslash character may only be included by using an escape sequence.

7.6 Notes

Note 1

The SysML v2 example shown in this Test Case works in the SysMl v2 Pilot implementation, however the JupyterLab visualisation with `%viz` appears unable to handle unrestricted names:

```
%viz --view=tree 'My PackageVehicles'

ERROR:Couldn't resolve reference to Element ''My'
```

8 Test Case 03-001: *AttributeDefinition* element

8.1 Description

Explores characterists of *AttributeDefinition* element.

8.2 Scope

The scope of this Test Case is uSysML v0.03, and the applicable keywords are: `package`, `part def`, and `part`, `attribute def`, and `attribute`.

8.3 SysML v2 textual notation

```
// 'attribute def' within the sysml file scope, outside any packages
attribute def ModuleId;

package PackageVehicles {

    // 'attribute def' within a top level package
    attribute def PackageVehiclesID;

    package VehicleAccessories {

        // 'attribute def' within a nested package
        attribute def AccessoryID;

        package SeatCover {
            // 'attribute def' within a second nested package-- no limit to possible nesting
            attribute def SeatCoverColor;
        }
    }

    part def Vehicle {
        // 'attribute def' within 'part def'
        attribute def Color;
    }
```

```
    part def Wheel;

    part vehicle:Vehicle {
        // 'attribute def' within 'part'
        attribute def RegistrationNumber;
        part w:Wheel[4];
    }
}
```

8.4 uSysML output

```
Root.ModuleId [AttributeDef]
Root.PackageVehicles [Package]
 Root.PackageVehicles.PackageVehiclesID [AttributeDef]
 Root.PackageVehicles.VehicleAccessories [Package]
  Root.PackageVehicles.VehicleAccessories.AccessoryID [AttributeDef]
  Root.PackageVehicles.VehicleAccessories.SeatCover [Package]
   Root.PackageVehicles.VehicleAccessories.SeatCover.SeatCoverColor [AttributeDef]
 Root.PackageVehicles.Vehicle [PartDef]
  Root.PackageVehicles.Vehicle.Color [AttributeDef]
 Root.PackageVehicles.Wheel [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.RegistrationNumber [AttributeDef]
  Root.PackageVehicles.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles.Wheel
```

8.5 Discussion

Placeholder.

8.6 Notes

Placeholder.

9 Test Case 03-002: *Attribute* element

9.1 Description

Explores characterists of *Attribute* element.

9.2 Scope

The scope of this Test Case is uSysML v0.03, and the applicable keywords are: `package`, `part def`, and `part`, `attribute def`, and `attribute`.

9.3 SysML v2 textual notation

```
// 'attribute def' within the sysml file scope, outside any packages
attribute def ModuleId;

package PackageVehicles {

    // 'attribute def' within a top level package
    attribute def PackageVehiclesID;

    package VehicleAccessories {

        // 'attribute def' within a nested package
        attribute def AccessoryID;
```

```
        package SeatCover {
            // 'attribute def' within a second nested package-- no limit to possible nesting
            attribute def SeatCoverColor;
        }
    }

    part def Vehicle {
        // 'attribute def' within 'part def'
        attribute def Color;
    }

    part def Wheel;

    part vehicle:Vehicle {
        // 'attribute def' within 'part'
        attribute def RegistrationNumber;
        part w:Wheel[4];
    }
}
```

9.4 uSysML output

```
Root.ModuleId [AttributeDef]
Root.PackageVehicles [Package]
 Root.PackageVehicles.PackageVehiclesID [AttributeDef]
 Root.PackageVehicles.VehicleAccessories [Package]
  Root.PackageVehicles.VehicleAccessories.AccessoryID [AttributeDef]
  Root.PackageVehicles.VehicleAccessories.SeatCover [Package]
   Root.PackageVehicles.VehicleAccessories.SeatCover.SeatCoverColor [AttributeDef]
 Root.PackageVehicles.Vehicle [PartDef]
  Root.PackageVehicles.Vehicle.Color [AttributeDef]
 Root.PackageVehicles.Wheel [PartDef]
 Root.PackageVehicles.vehicle [PartUsage]
    typed by=Root.PackageVehicles.Vehicle
  Root.PackageVehicles.vehicle.RegistrationNumber [AttributeDef]
  Root.PackageVehicles.vehicle.w [PartUsage]
      multiplicity=4
      typed by=Root.PackageVehicles.Wheel
```

9.5 Discussion

Placeholder.

9.6 Notes

Placeholder.