<config>  
<output path=’C:\Users\ndavis\git\OnfInfoModelOutput\ModelDescriptions\TR-512.3\_OnfCoreIm-Foundation.docx' />  
</config>

<context model=’C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreModel.uml' element=’{0}’ importedBundles='gmf;papyrus' searchMetamodels='true'/>

<gendoc><drop/>

Change path substrings above from “{path for output files}\” to your local path for the output files and “{path for CoreModel}\” to your local path for the Core Model. <drop/>

DELETE: Prior to publishing this –gd.docx (including for review), change path substrings above from “C:\Users\ndavis\git\OnfInfoModelOutput\” to “{path for output files}\” and from “C:\Users\ndavis\git\ONFInfoModel\OnfModel\” to “{path for CoreModel}\” <drop/>



Core Information Model (CoreModel)

TR-512.3

Foundation

(Identifiers, naming and state)

Version 1.4

November 2018

ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.4

**Disclaimer**

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation  
1000 El Camino Real, Suite 100, Menlo Park, CA 94025  
[www.opennetworking.org](http://www.opennetworking.org)

©2018 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

**Important note**

This Technical Recommendations has been approved by the Project TST, but has not been approved by the ONF board.  This Technical Recommendation is an update to a previously released TR specification, but it has been approved under the ONF publishing guidelines for ‘Informational’ publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents.  The designation of ‘-info’ at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

Finalizing this document once generated… delete this text prior to publication:

* Replace “{{..}}” with square brackets (which trip up Gendoc)
* Select text in document from beginning of table of contents (first line) to end of document
  + Click menu item “Update Field” (on this large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
  + Repeat “update fields” 2 more times (on the same large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
* Remove reviewer comment

Note that the table of contents and figures need to be updated several times as the table length changes the page numbering and the cross references will need to be re-updated.

Table of Contents

[Disclaimer 2](#_Toc434403079)

[Open Networking Foundation 2](#_Toc434403080)

[Document History 5](#_Toc434403081)

[1 Introduction 5](#_Toc434403082)

[2 References 6](#_Toc434403083)

[3 Definitions 8](#_Toc434403084)

[3.1 Terms defined elsewhere 8](#_Toc434403085)

[3.2 Terms defined in this TR 8](#_Toc434403086)

[4 Abbreviations and acronyms 8](#_Toc434403087)

[5 Conventions 10](#_Toc434403088)

[5.1 UML modeling conventions 10](#_Toc434403089)

[5.2 Lifecycle Stereotypes 10](#_Toc434403090)

[5.3 Diagram Keys 11](#_Toc434403091)

[6 Overview of the CoreModel Fragment 13](#_Toc434403092)

[6.1 Overview of the CoreNetworkModel of the CoreModel 15](#_Toc434403093)

[6.1.1 LogicalTerminationPoint (LTP) and LayerProtocol (LP) 17](#_Toc434403094)

[6.1.2 ForwardingDomain (FD) 18](#_Toc434403095)

[6.1.3 ForwardingConstruct (FC), EndPoint (EP), FcRoute and FcSwitch 18](#_Toc434403096)

[6.1.4 Link and LinkEnd 18](#_Toc434403097)

[6.1.5 NetworkElement, NetworkControlDomain and SdnController 19](#_Toc434403098)

[6.2 CoreFoundationModel 19](#_Toc434403099)

[6.2.1 Naming and identifiers 19](#_Toc434403100)

[6.2.2 States 22](#_Toc434403101)

[6.2.2.1 Relationship between states in Provider context 22](#_Toc434403102)

[6.2.2.2 Relationship between states in the client and provider context 23](#_Toc434403103)

[6.3 Termination Fragment 24](#_Toc434403104)

[6.4 Forwarding Fragment 27](#_Toc434403105)

[6.4.1 Basic Forwarding 27](#_Toc434403106)

[6.4.2 Forwarding Construct Specification and other details of Forwarding 28](#_Toc434403107)

[6.5 Topology Fragment 32](#_Toc434403108)

[6.5.1 Basic Topology 35](#_Toc434403109)

[6.5.2 Topology and views 40](#_Toc434403110)

[6.5.3 View boundaries and intermediates 46](#_Toc434403111)

[6.5.4 More on views and names/identifiers 47](#_Toc434403112)

[6.5.5 Off-network reference and the clients view 50](#_Toc434403113)

[6.5.6 Physical Port reference 51](#_Toc434403114)

[6.5.7 Detailed properties of Topology 53](#_Toc434403115)

[6.6 Directionality 55](#_Toc434403116)

[7 Future CoreModel areas 60](#_Toc434403117)

[8 UML model files 60](#_Toc434403118)

[8.1 Papyrus File 60](#_Toc434403119)

[9 Data Dictionary 62](#_Toc434403120)

[9.1 Core Network Model data dictionary 62](#_Toc434403121)

[9.1.1 Classes 62](#_Toc434403122)

[9.1.1.1 {{cl.name/}} 62](#_Toc434403123)

[9.1.2 Data Types 65](#_Toc434403124)

[9.1.2.1 {{dt.name/}} 65](#_Toc434403125)

[9.1.3 Enumeration Types 67](#_Toc434403126)

[9.1.3.1 {{dt.name/}} 67](#_Toc434403127)

[9.1.4 Primitive Types 68](#_Toc434403128)

[9.1.4.1 {{dt.name/}} 68](#_Toc434403129)

[9.2 Core Foundation Model data dictionary 68](#_Toc434403130)

[9.2.1 Classes 68](#_Toc434403131)

[9.2.1.1 {{cl.name/}} 68](#_Toc434403132)

[9.2.2 Data Types 71](#_Toc434403133)

[9.2.2.1 {{dt.name/}} 71](#_Toc434403134)

[9.2.3 Enumeration Types 73](#_Toc434403135)

[9.2.3.1 {{dt.name/}} 73](#_Toc434403136)

[9.2.4 Primitive Types 74](#_Toc434403137)

[9.2.4.1 {{dt.name/}} 74](#_Toc434403138)

[9.3 Core Enhancements data dictionary 74](#_Toc434403139)

[9.3.1 Classes 74](#_Toc434403140)

[9.3.1.1 {{cl.name/}} 74](#_Toc434403141)

[9.3.2 Data Types 77](#_Toc434403142)

[9.3.2.1 {{dt.name/}} 77](#_Toc434403143)

[9.3.3 Enumeration Types 79](#_Toc434403144)

[9.3.3.1 {{dt.name/}} 79](#_Toc434403145)

[9.3.4 Primitive Types 80](#_Toc434403146)

[9.3.4.1 {{dt.name/}} 80](#_Toc434403147)

[9.4 Core Enhancement Fragments data dictionary 80](#_Toc434403148)

[9.4.1 Classes 80](#_Toc434403149)

[9.4.1.1 {{cl.name/}} 80](#_Toc434403150)

[9.4.2 Data Types 83](#_Toc434403151)

[9.4.2.1 {{dt.name/}} 83](#_Toc434403152)

[9.4.3 Enumeration Types 85](#_Toc434403153)

[9.4.3.1 {{dt.name/}} 85](#_Toc434403154)

[9.4.4 Primitive Types 86](#_Toc434403155)

[9.4.4.1 {{dt.name/}} 86](#_Toc434403156)

[10 Additional figures related to potential extensions 87](#_Toc434403157)

[10.1 State extensions 87](#_Toc434403158)

[10.2 LTP Specification 87](#_Toc434403159)

[10.3 Model structure rules 88](#_Toc434403160)

[11 Addendum Translation table 91](#_Toc434403161)

[12 Back matter 93](#_Toc434403162)

[12.1 Editors 93](#_Toc434403163)

[12.2 Contributors 93](#_Toc434403164)

List of Figures

[Figure 1-1 Methodology of IM and DS Development 6](#_Toc430780029)

Document History

| **Version** | **Date** | **Description of Change** |
| --- | --- | --- |
| 1.0 | March 30, 2015 | Initial version of the base document of the “Core Information Model” fragment of the ONF Common Information Model (ONF-CIM). |
| 1.1 | November 24, 2015 | Version 1.1 |
| 1.2 | September 20, 2016 | Version 1.2 {{Note Version 1.1 was a single document whereas 1.2 is broken into a number of separate parts}} |
| 1.3 | September 2017 | Version 1.3 {{Published via wiki only}} |
| 1.3.1 | January 2018 | Addition of text related to approval status. |
| 1.4 | November 2018 | No changes. |

# Introduction

This document is an addendum to the TR-512 ONF Core Information Model and forms part of the description of the ONF-CIM. For general overview material and references to the other parts refer to [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_OnfCoreIm-Overview.pdf).

## References

For a full list of references see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_OnfCoreIm-Overview.pdf).

## Definitions

For a full list of definition see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_OnfCoreIm-Overview.pdf).

## Conventions

See [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_OnfCoreIm-Overview.pdf) for an explanation of:

* UML conventions
* Lifecycle Stereotypes
* Diagram symbol set

## Viewing UML diagrams

Some of the UML diagrams are very dense. To view them either zoom (sometimes to 400%) or open the associated image file (and zoom appropriately) or open the corresponding UML diagram via Papyrus (for each figure with a UML diagram the UML model diagram name is provided under the figure or within the figure).

## Understanding the figures

Figures showing fragments of the model using standard UML symbols and also figures illustrating application of the model are provided throughout this document. Many of the application-oriented figures also provide UML class diagrams for the corresponding model fragments (see [TR-512.1](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\TR-512.1_OnfCoreIm-Overview.pdf) for diagram symbol sets). All UML diagrams depict a subset of the relationships between the classes, such as inheritance (i.e. specialization), association relationships (such as aggregation and composition), and conditional features or capabilities. Some UML diagrams also show further details of the individual classes, such as their attributes and the data types used by the attributes.

# Introduction to the Foundation Model

The focus of this document is the parts of Core Foundation Model of the ONF-CIM that deal with naming, identifiers and state.

The CoreFoundationModel covers all aspects of naming and addressing for all classes in the ONF-CIM. The focus of this document is:

* Definition of terminology related to naming and identity
* Description of the model for naming and identifiers that is inherited by other models
* Description of the state model

A data dictionary that sets out the details of all classes, data types and attributes is also provided ([TR-512.DD](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.DD_OnfCoreIm-DataDictionary.pdf)).

# CoreFoundationModel

This Model includes all aspects of the core that are relevant to all other parts of the ONF CIM such as identifiers, naming and states.

## Naming and identifiers

### Key considerations

To communicate about a thing it is necessary to have some way of referring to that thing, i.e., to have some reference. Terms such as name and identifier are often used when describing the reference. Unfortunately these terms in general usage have ambiguity in their definition that leads to erroneous system behavior. To ensure that the controller system behavior is not erroneous, the model will adopt the following principal definitions:

* Entity[[1]](#footnote-1): Has identity, defined boundary, properties, functionality and lifecycle in a global context.
  + Examples: An Equipment (see [TR-512.6](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.6_OnfCoreIm-Physical.pdf)), an LTP (see [TR-512.2](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.2_OnfCoreIm-ForwardingAndTermination.pdf))
* Entity Feature[[2]](#footnote-2): An inseparable, externally distinguishable part of an entity.
  + Examples: A pin in connector (see [TR-512.6](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.6_OnfCoreIm-Physical.pdf)), the ports of a FC (see [TR-512.2](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.2_OnfCoreIm-ForwardingAndTermination.pdf)), a face of a cube, the handle of a cup.
  + Note that this is important from a modeling perspective as the representation appears similar to that of an Entity
  + Represented using a UML class
* Role: A specific structure of responsibilities, knowledge, skills, and attitudes in the context of some activity or greater structure. The role has an identity and identifier.
* Identifier: A property of an entity/role with a value that is unique within an identifier space, where the identifier space is itself globally unique, and immutable. An identifier carries no semantics with respect to the purpose or state of the entity.
* Universally Unique Identifier[[3]](#footnote-3) (UUID): An identifier that is universally unique.
* Local ID: An identifier that is unique in the context of some scope that is less than the global scope.
* Name: A property of an entity with a value that is unique in some namespace but may change during the life of the entity. A name carries no semantics with respect to the purpose of the entity.
* Label: A property of an entity with a value that is not expected to be unique and is allowed to change. A label carries no semantics with respect to the purpose of the entity and has no effect on the entity behavior or state.
  + A label can be used to carry a freeform text string for any operator purpose. The contents of a label in one view may happen to be the value of a name or identifier in another view. From the perspective of the view with the label there is no expectation other than the value is a string.
* Place: Where something is located
* Address: A structure of named values[[4]](#footnote-4) in some address space that defines a location (a volume in that address space) where the structure is a nested hierarchy.
  + A named value may be a name or identifier, the name of the value may be a name or identifier
* Route: the way (via specified intermediate locations and paths) to get to one location from another.
* Property: A quality associated with a thing, structure or location.
* Semantics: Meaning.
* Reference: Data in a communication between two applications that allows a shared understanding of the individual things.
  + This could be an identifier (including a UUID), a name, an address, or a route, depending upon the needs

Note:

* An entity may be known to be at a place in some functional or physical structure.
* A role may be known to be at a place in some process or behavioral structure.

The figure below illustrates the naming/identifier-related attributes defined in the ONF-CIM. They are Universally Unique ID (UUID), Local ID, Name and Label.

The model includes two abstract classes that provide names and identifiers, the GlobalClass and the LocalClass.[[5]](#footnote-5) A GlobalClass represents a type of thing that has instances which can exist in their own right (independently of any others). A LocalClass represents a type of thing that is inseparable from a GlobalClass, but that is a distinct feature of that GlobalClass such that the instances of LocalClass are able to have associations with other instances. The mandatory LocalId of the LocalClass instance is unique in the context of the GlobalClass instance, from which it is inseparable.

The model also includes Extension which is not related to naming/identification. Extension provides an opportunity to define properties not declared in the class that extend the class enabling a realization with simple ad-hoc extension of standard classes to be conformant.

Note that a UUID is applicable only to global type object classes (i.e., subclass of GlobalClass) that their instances can exist on their own right, e.g., LTP, FD, Link, FC, and NetworkElement. The other naming/identifier-related attributes are applicable to both global type object classes and local type object classes (i.e., subclass of LocalClass).[[6]](#footnote-6)

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Foundation-CommonPackagesNoNotes’, ’Class Diagram for Naming and Identifier of Objects’)/]

[/for]<drop/>

### Classes and attributes

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘SuperClassesAndCommonPackages’))]<drop/>

#### [cl.name/]

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’SuperClassesAndCommonPackages’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/for]<drop/>

### DataTypes

[for (dt:DataType | DataType.allInstances()->sortedBy(name))]<drop/>

[if dt.oclIsTypeOf(DataType)]<drop/>

[if (dt.qualifiedName.contains(‘SuperClassesAndCommonPackages’))]<drop/>

#### [dt.name/]

Inserts the details of the data type in first quotes from the package in second quotes <drop/>  
[dt.insertDataType(dt.name,’SuperClassesAndCommonPackages’)/]

Inserts the attributes of the class <drop/>   
[dt.insertDataTypeAttributeTableBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### Use of names, identifiers and addresses

The following figure provides various examples of naming and identifier. The figure is currently under development. The figure includes diagram element from [TR-512.5](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.5_OnfCoreIm-Resilience.pdf) and [TR-512.6](file:///C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreGendoc\ModelDescriptions\TR-512.6_OnfCoreIm-Physical.pdf).



Figure 6-33 Sketch of names, identifiers and addresses for various entities

## States

The Core Foundation Model also defines a State\_Pac artifact, which provides state attributes. The work on states is preliminary at this stage (it is derived from {{ITU-T X.731}}). The State\_Pac is inherited by GlobalClass and LocalClass object classes. The use of these states provides a consistent way represent the overall operability, usability and current usage of the resource.

It should be noted that the states are «Preliminary»/«Experimental».

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘State-FullModel’, ’States for all Objects’)/]

[/for]<drop/>

### Classes and attributes

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘StateModel’))]<drop/>

#### [cl.name/]

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’StateModel’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/for]<drop/>

### Enumerations

[for (dt:DataType | DataType.allInstances()->sortedBy(name))]<drop/>

[if dt.oclIsTypeOf(Enumeration)]<drop/>

[if (not dt.qualifiedName.contains(‘CoreFoundationModel’))]<drop/>

[else] <drop/>  
[dt.insertEnums()/]

[/if]<drop/>

[else] [/if]<drop/>

[/for]<drop/>

### Relationship between states in Provider context

If the lifecycleState is PLANNED then the operationalState must be DISABLED and the administrativeState should be LOCKED.

In all other circumstances the states are independent.

### Relationship between states in the client and provider context

The table below lists the states in the provider context that influence the states in the client context. The client does not have direct visibility of the provider states but does perceive effect through the operationalState.

Table 1: Influence of Provider state on Client state

|  |  |  |  |
| --- | --- | --- | --- |
| **Provider State** | **Value** | **Client State** | **Value** |
| operationalState | DISABLED | operationalState | DISABLED |
| administrativeState | SHUTTING\_DOWN | operationalState | DISABLED |
| administrativeState | LOCKED | operationalState | DISABLED |
| lifecycleState | PLANNED | operationalState | DISABLED |
| lifecycleState | POTENTIAL\_BUSY | operationalState | DISABLED |

No other states in the client context have a dependency on the state in the provider context.

None of the states in the client context influence the states in the provider context.

The administrativeState in the provider context is not visible in the client context. The client context may maintain an independent administrativeState.

The provider controls the lifecycleState that is visible to the client context.

### State diagrams

These state diagrams are experimental sketches that will be refined in following releases.

#### Administrative State

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘State-AdministrativeState’, ’Administrative State’)/]

[/for]<drop/>

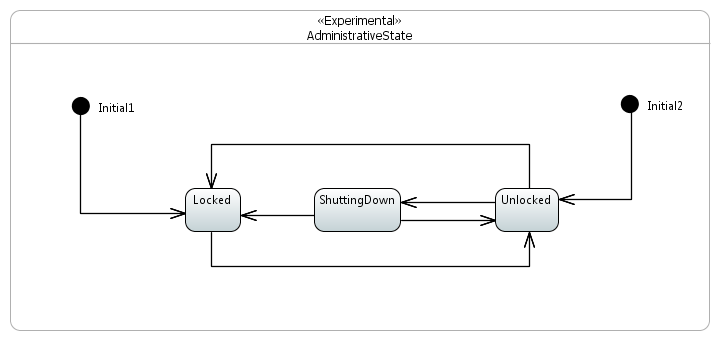


Figure 6-2 Administrative State

#### Operational State

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘State-OperationalState’, ’Operational State’)/]

[/for]<drop/>

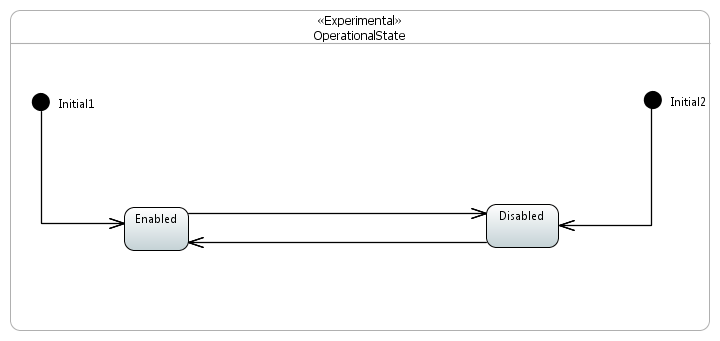


Figure 6-2 Operational State

#### Lifecycle State

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘State-LifecycleState’, ’Lifecycle State’)/]

[/for]<drop/>

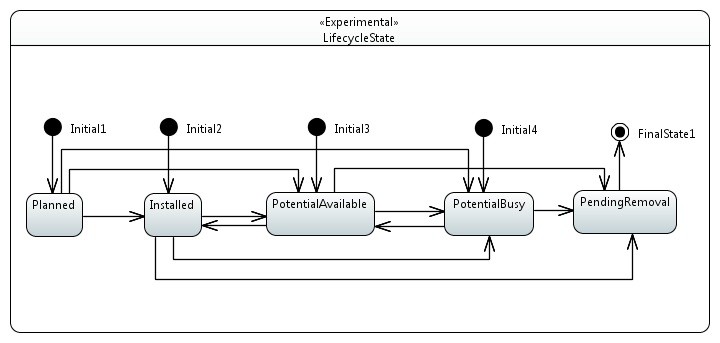


Figure 6-2 Lifecycle State

### Use of states

Examples to be added.

**End of document**

</gendoc><drop/>

To take latest template: <drop/>

* delete text from “Template version…” to end of file <drop/>
* insert a line in “Normal” style<drop/>
* insert text (Insert 🡪 Object 🡪 Text from File… (alt njf)) from: <drop/>
  + TR-512.GT\_OnfCoreIm-CommonGendocTemplate-Fragments.docx <drop/>

Template version 0.0.10 17 September 2017 <drop/>

# Fragment: Insert class <drop/>

<fragment name=’insertClass’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
<arg name=’className’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (not cl.qualifiedName.contains(packageName))]<drop/>  
[else] <drop/>  
[if(cl.name.contains(className))]<drop/>

Qualified Name: [cl.qualifiedName/]

[for (co:Comment | cl.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (cl.isAbstract)]<drop/>

This class is abstract.

[/if]<drop/>

[if (cl.oclAsType(uml::Class).general ->notEmpty())]<drop/>

Inherits properties from:

[for (gen:Class | cl.oclAsType(uml::Class).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>  
[if(not st.name.contains(‘OpenModelClass’))]<drop/>

This class is [st.name/].

[else] <drop/>  
[/if]<drop/>  
[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Insert standard diagram <drop/>

<fragment name=’insertStandardDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 6-2 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert small diagram <drop/>

<fragment name=’insertSmallDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 6-2 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief not Obsolete<drop/>

<fragment name=’insertAttributeRowBriefNotObsolete’ importedBundles=’commons;gmf;papyrus’><drop/>

Does not work unless we have Mature stereotype… <drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

[for (st:Stereotype | p.getAppliedStereotypes())]<drop/>

[if(not st.name.contains(‘OpenModelAttribute’))]

[if(not st.name.contains(‘Obsolete’))]

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief <drop/>

<fragment name=’insertAttributeRowBrief’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

</fragment><drop/>

# Fragment: Start attribute table brief <drop/>

<fragment name=’insertAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Attribute table brief <drop/>

<fragment name=’insertAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert Ten Specified Attribute table brief <drop/>

<fragment name=’insertTenSpecifiedAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

<arg name=’p1’ type=‘String’/><drop/>

<arg name=’p2’ type=‘String’/><drop/>  
<arg name=’p3’ type=‘String’/><drop/>  
<arg name=’p4’ type=‘String’/><drop/>  
<arg name=’p5’ type=‘String’/><drop/>  
<arg name=’p6’ type=‘String’/><drop/>  
<arg name=’p7’ type=‘String’/><drop/>  
<arg name=’p8’ type=‘String’/><drop/>  
<arg name=’p9’ type=‘String’/><drop/>  
<arg name=’p10’ type=‘String’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert DataType <drop/>

<fragment name=’insertDataType’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
<arg name=’dataTypeName’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (dt.qualifiedName.contains(packageName))]<drop/>  
[if(dt.name.contains(dataTypeName))]<drop/>

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits properties from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]<drop/>

[for (gen:Class | dt.oclAsType(uml::DataType).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>  
This class is [st.name/].

[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Start Data Type attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Data Type Attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertDataTypeAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
[if dt.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [dt.name/]

<table><drop/>

[dt.insertDataTypeAttributeTableHeader ()/]

[for (p:Property|dt.ownedAttribute)]<drop/>

[p.insertAttributeRowBrief ()/]

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert enums <drop/>

<fragment name=’insertEnums’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

#### [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>

Applied stereotypes:

[if dt.getAppliedStereotypes()->notEmpty()] <drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

* [st.name/]

[/for]<drop/>

[else] No stereotypes applied

[/if]<drop/>

[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits literals from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]

[/if]<drop/>

[if (dt.oclAsType(Enumeration).ownedLiteral->notEmpty())]<drop/>

Contains Enumeration Literals:

[for (e:EnumerationLiteral|dt.oclAsType(Enumeration).ownedLiteral)]<drop/>

* [e.name/]:
  + [for (co:Comment | e.ownedComment)]<drop/>
  + <dropEmpty>[cleanAndFormat(co.\_body.clean())/]
  + </dropEmpty>[/for]<drop/>
  + [if dt.getAppliedStereotypes()->notEmpty()] <drop/>
  + Applied stereotypes:
    - [for (st:Stereotype | e.getAppliedStereotypes())]<drop/>
    - [st.name/]
    - [/for]<drop/>
  + [/if]<drop/>

[/for]<drop/>

[/if]<drop/>

</fragment><drop/>

1. An Entity is represented using a UML class [↑](#footnote-ref-1)
2. A Feature of an Entity is represented using a UML class [↑](#footnote-ref-2)
3. The term GUID was used in the previous version of the model. The change is in recognition of the more generally applicability of UUID [↑](#footnote-ref-3)
4. A named value is simply a tuple with two terms, one being a value and the other being the name of that value. For example in a street address a value may be “London” and the name of that value would be “City”. [↑](#footnote-ref-4)
5. The model also provides ConditionalPackage to supply names and identifiers to \_Pac classes but this is currently experimental. [↑](#footnote-ref-5)
6. The intention is that only classes from the Core Model are shown in the figure. The classes shown are essentially illustrative. There is another figure in the model that captures Core Model inheritance in detail. All classes from all fragments should inherit from GlobalClass, LocalClass or ConditionalPackage. There is no issue with model dependency as the inheritance association is maintained with the class that is inheriting properties. Although not mandatory, it would seem advisable to maintain a figure per fragment that shows all classes from that fragment and their inheritance. [↑](#footnote-ref-6)