<config>  
<output path=’C:\Users\ndavis\git\OnfInfoModelOutput\ModelDescriptions\TR-512.6\_OnfCoreIm-Physical.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-512.6

Physical

Version 1.5

September 2021

ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.5

**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)

©2021 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 CoreNetworkModule 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 CoreFoundationModule 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 Module 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 Module 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 | Enhancements to connector model |
| 1.5 | September 2021 | Enhancements to model structure |

# Introduction to the document suite

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](../TR-512.1_OnfCoreIm-Overview.pdf).

## References

For a full list of references see [TR-512.1](../TR-512.1_OnfCoreIm-Overview.pdf).

## Definitions

For a full list of definition see [TR-512.1](../TR-512.1_OnfCoreIm-Overview.pdf).

## Conventions

See [TR-512.1](../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 as well as 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](../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 Physical model

The focus of this document is the modeling of physical things, especially equipment, in the ONF-CIM.

Note:

* A majority of the Physical model is experimental at this stage (some key classes and attributes have been upgraded to Mature/Preliminary since V1.2). It was considered vital to publish the work in progress on equipment as it is clearly an important part of the overall model. Many of the attributes and classes are not fully documented in the model.
* The Physical model deals with physical things where a physical thing is something that can be “measured with a ruler”[[1]](#footnote-1)

This document:

* Introduces the Physical model structure
* Describes the key classes of the Physical model
* Explains the attributes of the Physical model
* Describes the relationship between the Connector and the LogicalTerminationPoint (LTP)
* Shows how the model deals with the relationship between physical and functional views
* Explains how the specification model describes equipment schemes (rules etc)
* Highlights work in progress to further advance the Physical model

The Physical model relates to:

* The Core Network Model including Termination and Forwarding described in [TR-512.2](TR-512.2_OnfCoreIm-ForwardingAndTermination.pdf) and Topology described in [TR-512.4](TR-512.4_OnfCoreIm-Topology.pdf).
* The generalized processing and constraint model described in [TR-512.11](TR-512.11_OnfCoreIm-ProcessingConstruct.pdf)
* The specification model described in [TR-512.7](TR-512.7_OnfCoreIm-Specification.pdf).

A data dictionary that sets out the details of all classes, data types and attributes is also provided ([TR-512.DD](TR-512.DD_OnfCoreIm-DataDictionary.pdf) ).

# Physical model detail

This section starts with a basic view of the equipment classes, then progresses through detail to sophisticated (and highly experimental) representations of equipment model constraints.

## Equipment Pattern

The figure below sets out the basic equipment pattern.

The classes of the model are described briefly after the figure. The associations are assumed to be sufficiently self-explanatory at this stage.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-Pattern’, ’Skeleton Class Diagram of key object classes’)/]

[/for]<drop/>

Taking a simple chassis, pictured in the figure below, as an example, we can consider the Holders to be the spaces within a piece of Equipment where a Holder is designed to accommodate another piece of Equipment. In a normal chassis it is possible that an Equipment may occupy several Holder, however it is not possible for a Holder to accommodate more than one Equipment at a time.



Figure 6-33 - Equipment and Holder example

There are two distinct roles for the Equipment entity controlled by the one attribute shown in the model figure above (isFieldReplaceable):

* Field Replaceable Unit (FRU):
  + Can be replaced in the field.
    - May be standalone
    - May plug in to a Holder in another Equipment (if not stand-alone)
* Non-Field Replaceable Unit (NFRU):
  + Cannot be replaced in the field. Is simply a subordinate part of an FRU (or another NFRU – where there must be an FRU at the top of the hierarchy).
  + Does not have any exposed Holders (any associated Holders are assumed to belong to the containing FRU).
  + Does not have any Connectors (any associated Connectors are assumed to belong to the containing FRU).

A method for representation of these restrictions is covered in section 6.6 FRU and non-FRU on page 9.

Connectors allow for Cables to be plugged in. So, for example, an SFP (Small Form-factor Pluggable) is a piece of Equipment that plugs in a Holder, and has a Connector on its front for a Cable with a Connector to be plugged in.

Looking at the picture below, we have removed some of the chassis panel to show the inside. We see that the circuit-pack (Equipment) will actually plug into Connectors on the (non-FRU) backplane. If we wish to explicitly represent the Connectors on the backplane, then we use association HolderEncapsulatesConnector to relate the Connector to the Holder it is at the back of.



Figure 6-33 – Inside the Equipment

### Equipment

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

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

[if (cl.name.contains(‘Equipment’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_containedHolder’, ‘\_addressedByHolder’, ‘\_encapsulatedNonFru’, ‘\_connector’, ‘\_exposedCable’, ‘isFieldReplaceable’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### Holder

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

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

[if (cl.name.contains(‘Holder’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_occupyingFru’, ‘\_connector’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### Connector

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

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

[if (cl.name.contains(‘Connector’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_connector’, ‘role’, ‘orientation’, ‘connectorType’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### Cable

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

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

[if (cl.name.contains(‘Cable’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_connector’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

## Equipment Detail

The figure below highlights classes that group together related attributes (related as suggested by the name of the class). As noted in the key to the diagram the attributes are also grouped on the degree of variation. This latter grouping will guide the construction of specifications indicating what can reside only in the spec and what has to be available per instance (see [TR-512.7](TR-512.7_OnfCoreIm-Specification.pdf) for more information)

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-DetailWithoutAttributes’, ’Equipment Detail Structure’)/]

[/for]<drop/>

### Invariant Equipment Detail

The following classes have attributes that do not change in value for the life of the Equipment. Some are the same value across all Equipment of the same type.

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

[if (not cl.qualifiedName.contains(‘InvariantDetail’))]<drop/>

[else] <drop/>

#### [cl.name/]

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

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

[/if]<drop/>

[/for]<drop/>

### Dynamic Equipment Detail

The following classes have attributes that can change in value during the life of the Equipment.

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

[if (not cl.qualifiedName.contains(‘DynamicDetail’))]<drop/>

[else] <drop/>

#### [cl.name/]

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

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

[/if]<drop/>

[/for]<drop/>

## Connector to LTP Model

The figure in this section relates the physical aspects of the model at the boundary (connector, pin etc) to the functional aspects at the boundary (LTP, FC etc).

### The AccessPort

The AccessPort is the most abstract level of the point-oriented aspect of the physical model and is a bridge between the point aspects of the functional model and the physical model.

Consider the figure below (where physical things are shown in orange and functional in green). As shown in the figure, the Connector, introduced earlier, is decomposed into pins, hence the connector is a PinGroup where the pins are owned by the connector (and are not meaningful by themselves). A pin is not meaningfully divisible. The pins are laid out in the connector and hence have position in the connector.

Examining the functional aspects of the figure, the LTP is a SignalReferencePoint which may carry 1 or more ElementalSignals. An ElementalSignal may require more than one Pin and a Pin may support more than one ElementalSignal. In a simple case, an LTP may process only one ElementalSignal and that Signal may be supported by a single Pin that is by itself in a Connector. In this simple case there is clearly a 1:1 relationship between the LTP and the Connector.

However, this is a corner case. In a majority of, cases, the LTP carries more than one ElementalSignal and the relationship between the LTP and the Pins of the Connectors is not simple. Indeed, several signals may share, and be dependent on a single Pin or on several Pins. The groupings are such that if any pins were to be removed from the group, then all the signals of the group would be lost, and such that the signals supported by the PinGroup are indivisible when they flow through the pins.

To enable the necessary flexibility the Pins of the Connector are decoupled from the Signals of the LTP via the AccessPort. The AccessPort is a PinGroup where the Pins in the group may be scattered across several Connectors. The PinGroup is such that a group of LTPs depend upon “all” Pins in the group, i.e. if any pin is removed this would cause failure in signals such that the operation of all of the set of LTPs would be impacted (fail or degrade in some way).

The notes shown on the figure touch on some ongoing discussions.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ConnectorPinPortAndLTP’, ’Connector to LTP’)/]

[/for]<drop/>

#### Changes since 1.3.1

The definitions of some of the classes have been corrected. As a result:

* The GroupOfPins class has been eliminated from the model (as PinGroup is the GroupOfPins)
* The AccessPort is a PinGroup (rather than the other way round – an error)
* The SignalRefPtGroup relates to the AccessPort rather than the PinGroup

### The MultipleStrandSpan

The previous section dealt with the point aspect of the model. The relationship between the FC and the Cable is equivalent to the relationship between the LTP and the Connector.

The MultipleStrandSpan is the most abstract level of the arc-oriented aspect of the physical model and the bridge between the arc aspects of functional model and the physical model.

Consider the figure below (where physical things are shown in orange and functional in green). As shown in the figure, the Connector, introduced earlier, is decomposed into pins. The details of this aspect are covered in the previous section. The Pin is at the end of a Strand and the Connector at the end of a Cable.

It should be noted that a Cable may not have connectors at all/any ends and a Strand may not have Pins at all/any ends. The Cables and Strands of most interest in this model are the ones with Connectors and Pins respectively. Most of the other Cables/Strands are abstracted[[2]](#footnote-2).

Where the Strand does not have Pins, it may be spliced (the model only supports simple splice cases), or simply aggregated into a group of adjacent Strands which another Strand abstracts. The assumption here is that the detail of the sequence of Strands is not relevant, but it may be relevant to know that the apparent Strand between two points is made of a sequence of Strands of different properties. Where it is relevant to understand the actual sequence the \_splicedStrand attribute can be used to chain the Strands.

A Cable may have multiple strands and hence it is a GroupOfStrands where the Strands are owned by the Cable. A Strand is not meaningfully divisible. The Strand layout in the Cable is not considered relevant here, although it is expected that there may be some properties on a Cable that defines some aspect of Strand layout, e.g, twisted, coax. As a Cable can bundle Cables each internal segment of the Cable could be modelled if beneficial.

The elemental aspects of the FC are driven by the LTP. Clearly an FC can be decomposed into parts in the functional model. Here the relationship is simply from any FC to a GroupOfStrands that support the FC. The specific group is call the MultipleStrandSpan.

Examining the functional aspects of the figure in the previous section and considering a simple case, an FC may process only one ElementalSignal and that Signal may be supported by a single Strand that is by itself in a Cable. In this simple case there is clearly a 1:1 relationship between the FC and the Cable.

However, as noted earlier, this is a corner case. In a majority of cases, the FC carries more than one ElementalSignal and the relationship between the FC and the Strands of the Cable is not simple. Indeed, several signals may share, and be dependent on a single Strand or on several Strands. The groupings are such that if any Strands were to be removed from the group, then all the signals of the group would be lost, and such that the signals supported by GroupOfStrands are indivisible when they flow through the Strands.

To enable the necessary flexibility the Strands of the Cable are decoupled from the FC via the MultipleStrandSpan. The MultipleStrandSpan terminates on AccessPorts. The Strands in the group may be scattered across several Cables related to the way the Pins are scattered across several Connectors. However, it is likely that the Cable bundling of Cables may significantly reduce the spread of FCs across Cables compared to the spread of LTPs across Connectors (a Cable may have many Connectors and a Cable my bundle Cables). This means that the FCs carrying an indivisible signal will often travel along a single Cable through a single duct[[3]](#footnote-3).

The GroupOfStrands is such that the FC depend upon “all” Strands in the group. An FC may itself be a grouping of FCs such that if any Strand is removed this would cause failure in signals such that the operation of the grouping FCs would be impacted (fail or degrade in some way).

The notes shown on the figure touch on some ongoing discussions.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ConnectorPinCableStrandAndFc’, ’Cable to FC’)/]

[/for]<drop/>

### AccessPort and MultipleStrandSpan

The following figure shows both aspects of the model.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ConnectorPinCableStrandLtpAndFc’, ’Connector/Cable to LTP/FC’)/]

[/for]<drop/>

### Detailed description of the model fragment

The following sections list the classes introduced in this section and not covered in the previous sections of this document. See section 6.9 Physical port reference on page 11 for some pictorial examples of the interaction of some of the entities discussed here.

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

[if (not cl.qualifiedName.contains(‘ConnectorAndPin’))]<drop/>

[else] <drop/>

#### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## Equipment to Function Sketch

As ProcessingConstruct (see [TR-512.11](TR-512.11_OnfCoreIm-ProcessingConstruct.pdf)) was added in release 1.3 of the model and the Equipment to Function model was updated. Further analysis have improved some aspects of this model.

### Equipment to Function using ProcessingConstruct and ConstraintDomain

In principle, functionality is emergent from equipment when it is powered. In the model, there are several representations of functionality (PC, LTP etc). Understanding the relationship between relevant network functions and the physical environment is relevant for many management-control activities including network planning, fault localization and engineering works impact analysis.

The following figure shows the use of PC to model the functionality emergent from an Equipment assembly.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ProcessingConstructAndResilience’, ’Equipment to Function using ProcessingConstruct and showing Resilience’)/]

[/for]<drop/>

The axis of emergent functionality are the PC and CD. As shown in the figure above, a CD can be used to define the functional boundary of one[[4]](#footnote-4) or more equipments (EquipmentFunctionalBoundaryRepresentedByCd). At the base of relevant functional emergence is a PC that relates to one single equipment (PcEmergentFromRunningEquipment). PCs may be assembled into systems of PCs interconnected through their PcPorts (PcPortBoundToPcPort) that can then be viewed as PCs (PcIsAssemblyOfPc). A PC can also be broken down into alternative subordinate parts (also via PcIsAssemblyOfPc).

Hence, the ProcessingConstruct is used to represent:

* The function blocks supported by an Equipment
* The decomposition of each function block into atomic parts
* The assembly of the atomic parts into more aggregate functions (across Equipment boundaries potentially) that can be seen as function blocks for further decomposition and aggregation as appropriate

At some point in this complex recursion a relevant network function is emergent (e.g. LtpIsEmergentFromPc[[5]](#footnote-5)).

The “PcIsAssemblyOfPc” is essentially a compact form of the Component-System pattern (see [TR-512.A.2](TR-512.A.2_OnfCoreIm-Appendix-ModelStructurePatternsAndArchitecture.pdf)).

### “Equipment Protection”

A combination of Equipment can offer protected functions. This is not protection of the equipment as the individual Equipments are not in themselves physically resilient. It is also not protection of the entire capability of an equipment as some functions on an Equipment may be protected by functions on another equipment in one way, some in another way and some may not be protected.

The model enables the representation of the necessary degrees of protection. The essential protection is supported in terms of the PC where a PC instance may represent a function on one equipment, another PC instance may represent the same function on another equipment and a further PC instance represents the resilient function resulting from the appropriate combination of the two.

The model allows different levels of precision:

1. Basic representation via PcIsAssemblyOfPc such that a resilient PC is an assembly of non-resilient PCs
2. More explicit representation via PcPortBoundToPcPort where the functionality to be protected is exposed at a port of an instance of a PC on one equipment and at a port of an instance of the same type of PC on another equipment and these two ports are bound to ports of a PC that exposes a resilient form of the function function (i.e., the 0..2 cardinality of PcPortBoundToPcPort as shown in the previous figure above).
3. Detailed and generalized via PcPortBoundToFcPort where the same functions noted in (2) are not directly bound but are bound via an FC that allows exposure of the FcSwitch and Casc and other resilience properties in a standard way. This model can be used for complex protection schemes

The figure below shows two instances of function A on different equipments being combined to form a resilient function Ar.



Figure 6-35 “Equipment Protection”

See also Figure 6-35 Simplified sketch of forms of resilience in an NE.

### Classes associated with Equipment to function mapping with resilience

The following is a summary of the key classes and attributes related to Equipment to function mapping with resilience.

#### ProcessingConstruct (PC)

This section focuses on the aspects that support functional decomposition, functional aggregation/assembly and functional abstraction.

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

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

[if (cl.name.contains(‘ProcessingConstruct’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_equipment’, ‘\_composedPc’, ‘\_resilienceSelector’, ‘\_pcPort’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

#### PcPort

This section focuses on assembly of components with no need for LTPs where the component association in the assembly is direct and indirect via a resilience mechanism.

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

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

[if (cl.name.contains(‘PcPort’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_pcPort’, ‘\_fcPort’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

#### ForwardingConstruct (FC)

This section focuses on the protection aspects of the FC.

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

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

[if (cl.name.contains(‘ForwardingConstruct’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_fcPort’, ‘\_fcSwitch’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

#### FcSwitch

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

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

[if (cl.name.contains(‘FcSwitch’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_selectedFcPort’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

#### Equipment

This section focuses on physical support for functionality.

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

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

[if (cl.name.contains(‘Equipment’))]<drop/>

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

Inserts the attributes of the class <drop/>   
[cl.insertTenSpecifiedAttributeTableBrief (‘\_equipmentFunctionalBoundary’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘10’)/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### V1.4 refinements of V1.3 model

In release 1.3 there were two direct associations between PC and Equipment. These associations have been normalized with those from the Software model (see [TR-512.12](TR-512.12_OnfCoreIm-Software.pdf))

These have been replaced by a, more versatile but less apparent, use of ConstraintDomain.

### Obsolete model from V1.2 use to illustrate the V1.3 model

The figure below is a sketch model relating Equipment to Function from V1.2. Whilst the classes are obsolete they still provide a helpful illustration of the model. As the comment in the figure says, “All functions are essentially ProcessingConstruct” (i.e. any class with the word “function” in its name is replaces by PC in the V1.3 model described above). The Equipment functionality support is first exposed as coarse FunctionBlocks (e.g. arithmetic process, traffic process etc). The function block may be made resilient via a complex protection switch which can select the functionality from one or more instances[[6]](#footnote-6). The FunctionBlock is then decomposed into AtomicFunctions which can then be assembled to form AggregateFunctions (e.g. the LTP). These two steps allow versatile mapping from the hardware-oriented function blocks to the conceptual functions such as LTP and LayerProtocol where the conceptual function may be “smeared” across several FunctionBlocks. The model also allows for recursive decomposition and assembly to any depth to allow for cases where intermediate representations are necessary to describe the functional emergence. For some illustrative figures of this see section 7.1 Physical to functional model on page 13.

CoreModel diagram: Equipment-ObsoleteEquipmentToFunction

Figure 3-10 Equipment to Function

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

[if (not cl.qualifiedName.contains(‘EquipmentToFunction’))]<drop/>

[else] <drop/>

#### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## FRU and non-FRU

Note that the approach in this subsection to representation of the constraints using inheritance with stereotype is highly experimental. It is likely that this approach will change as the model progresses.

Considering the distinction between FRU and nonFRU there were two choices, model the FRU explicitly (and then expect instantiation of instances of distinct FRUs and non-FRUs classes) or model the FRU and non-FRU as Equipment with solely an attribute to indicate the case. The attribute approach was preferred but that lost rule detail present in the distinct class case.

So as not to lose the constraints, the FRU/non-FRU class distinctions were kept, those classes were made abstract and then applied as shown to the attribute based model developing an experimental technique

This technique is to use generalization modulated with stereotypes to represent the narrowing of a class to cover a defined case. The narrowed class does not gain attributes, the general class is fully populated whereas in the narrowed class attributes take specific fixed values where the specializations are all abstract and the generalization is concrete. The aim is to develop machine interpretable rule systems that allow the behavior of an instance of a generalized class to be constrained based upon the case.

The experimental stereotype « ClassificationRule » carries the properties that define the case, the stereotype « AbidesByRule » identifies the generalized association that constrained by the specialized association.

This model is experimental and requires significant further development. It is likely that an alternative form will eventually be used.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-EquipmentToHolderRules’, ’FRU and Non-FRU rules’)/]

[/for]<drop/>

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

[if (not cl.qualifiedName.contains(‘FruNonFruRules’))]<drop/>

[else] <drop/>

### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## Connector Rules

Note that the approach in this subsection to representation of the constraints using inheritance with stereotype is highly experimental. It is likely that this approach will change as the model progresses.

Similar to the previous section, the figure below shows an experimental method for representation of restrictions in the model. The figure below shows a representation of the rules for the exposure of Connectors on an Equipment accounting for both the FRU/Non-FRU differences and also for the differences between Connectors related to a Holder versus visible Connectors available to connect cables to.

Essentially the figure shows that Connectors related to Holders have different lifecycles and visibilities than Connectors related to cables.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ConnectorRules’, ’Connector rules’)/]

[/for]<drop/>

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

[if (not cl.qualifiedName.contains(‘ConnectorRules’))]<drop/>

[else] <drop/>

### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## Expected and Actual

This model fragment explores a representation of expected and actual where the Equipment pattern is augmented with composed parts representing the expected and actual settings.

The ExpectedEquipment is likely to be in terms of constraints, with many “don’t care” values (such as serial number) but is assumed to have a single precise definition of type and of position with respect to the Equipment/Holder it is to be contained in[[7]](#footnote-7).

The assumption is that when there is a mismatch, the expected Equipment will have a set of expected Holders and the actual Equipment a potentially overlapping set of actual Holders. The hierarchy is driven by Holder position.

Where the Equipment is stand-alone (not in a Holder) it is assumed that geographical location or other statement of place will enable detection of expectation/actual mismatch. Hence stand-alone Equipment can also have an expected and actual value. Also see section 7.2 Actual v expected on page 16 for a pictorial example of work in progress.

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

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘Equipment-ExpectedAndActual’, ’Expected and actual’)/]

[/for]<drop/>

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

[if (not cl.qualifiedName.contains(‘ExpectedAndActual’))]<drop/>

[else] <drop/>

### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## Specification

The figure below provides a fragment of a candidate specification model for Equipment, focusing on Holder compatibility and supported non-FRUs.

Note that the EquipmentEncapsulatesNonFru association and the HolderOccupiedByEquipment association (both shown in red) are essentially governed by the rules stated in the supportedNonFruType class and SupportedEquipmentType class (and their associated classes) respectively.

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

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

[/for]<drop/>

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

[if (not cl.qualifiedName.contains(‘EquipmentSpecification’))]<drop/>

[else] <drop/>

### [cl.name/]

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

[/if]<drop/>

[/for]<drop/>

## Physical Connector and conceptual Port

The Connector and Port are modelled as distinct abstract entities. It is likely that in both cases only the “name” is required in an instance realization. Both the Port and Connector represent fixed rules of grouping and these rules could be completely contained in the spec. The Connector also has physical properties but again these are fixed and could be contained in the spec.

An LTP instance provides an AccessPort reference. The AccessPort represents a PinGroup that relates to the Connectors via Pins as described in the appropriate spec. The Connector is part of an Equipment. An Equipment instance references a spec that identifies the Connectors.

There are many potential arrangements of association of LTP to AccessPort, Pin and Connector. The following sequence of figures provides a view of some of the variety.

As noted previously the AccessPort is a grouping of pins related to some coherent traffic flow. As can be seen from the sequence of figures there is no fixed relationship from AccessPort to Connector or Pin.



Figure 6-33 Basic cases of Access Port Reference

The figure above shows some simple LTP – AccessPort – Pin – Connector cases. The diagram case on the left could be where a single fiber is being used to convey a bidirectional signal (a coupler/splitter is within the Equipment) and hence only a single pin is required on a single Connector. In this case the Connector, Pin, AccessPort and LTP all have multiplicity {{1}}. The diagram case on the right could be again an optical case (with one pin per Connector) where the LTP is being considered as bidirectional but there are separate dedicated Equipments for each direction of traffic.

The pin of the Connector is essentially omni-directional (i.e. the media has no directional limitation and the material would allow the photons/electrons etc to propagate in any direction – see documentation on media in [TR-512.2](TR-512.2_OnfCoreIm-ForwardingAndTermination.pdf) and [TR-512.A.4](TR-512.A.4_OnfCoreIm-Appendix-AnalogueAndMediaExamples-L0.pdf)). The construction of the Connector (and the associated Strand of material) does in some cases provide such a narrow channel that the propagation is limited to essentially two directions but even in these cases the material is essentially omni-directional. The directionality designation associated with a Connector is “inherited” from the directionality of the termination function attached. In the figure above/below the rx and tx designations result from the attached terminations.



Figure 6-34 More Complex cases of intertwined Connectors

The diagram on the left in the figure above could represent a case where there is ribbon cable with multiple (in this case four) fibers terminated with one Connector and where each fiber is being used for only one direction of signal (but the fiber is inherently omni-directional as noted above). The two LTPs shown are both bidirectional and hence use two Pins each. As the signal is bidirectional in nature the AccessPort is also bidirectional



Figure 6-35 Unidirectional Cases

Other cases to consider:

* Multiple LTPs per AccessPort (as shown in the left and right cases in the diagram above)
* Multiple AccessPorts per pin

# Work in progress (see also [TR-512.FE](TR-512.FE_OnfCoreIm-FutureEnhancements.pdf))

## Addressing

Traditionally ports have been identified using addressing schemes based on physical positioning, however there are challenges related to the complexity of potential spread across Equipments and ports as discussed in the previous section. The figure below discusses some cases. This work needs to be taken further.



Figure 6-35 Connector/Port based addressing of LTPs

## Physical to functional model

The figures in this section highlight aspects of the physical to functional relationship. The physical to functional model has been enhanced in V1.3 as a result of the addition of the Processing Construct (see [TR-512.11](TR-512.11_OnfCoreIm-ProcessingConstruct.pdf)). As a result, the cases set out below can now be supported with the model. The specific examples have not yet been included in the detailed documentation. As a consequence this section has not yet been removed.

The figure below shows a simplified hierarchy of functions emerging from running hardware/equipment where the emergent behavior considered as “function services”. The figure is a rough sketch.



Figure 6-35 Simplified sketch of physical to functional

The figure below shows, very roughly, the emergence of various aggregate functions from functional blocks on two FRUs. In the figure:

* The FRUs are at the bottom (in brown)
* The inherent capabilities of the FRUs are shown as dotted boxes in the FRUs
* The emergent functional blocks are shown in green as are the aggregate functions
* A somewhat abbreviated progression from functional block to aggregate function via atomic function (not shown) to LTP (in blue at the top of the page) is shown
* The yellow bars represent different forms of protection essentially as FCs
* The functions in the yellow dotted shape are essentially virtual functions as the position of realization of the function is not fixed (and need not be known)



Figure 6-35 Simplified sketch of forms of resilience in an NE

## Actual v expected

The figure below shows various cases of actual-expectation mismatch.



Figure 6-35 Expectation v actual showing mismatch and blocking

This figure above shows a representation of a SUBRACK (Equipment) with 9 SLOTS (Holders). Each Holder has a dotted line down the middle where the symbols in the slot to the left of the dotted line represent equipping expectation and to the right represent actual equipping. The large colored blocks represent the CIRCUIT\_PACKs in the SLOTs (SLOT 8 is empty). The lower small colored rectangles represent the SLOTs in the CIRCUIT\_PACKs (A – E) and the SMALL\_FORMFACTOR\_PLUGGABLEs (Equipment) in these SLOTS.

Considering the slots in turn

* Slot 1
  + Expectation: Double slot card with subslots (Holders) A-C (where A and B are expected to be equipped
  + Actual: Double slot card of different type with subslots D & E where both are actually equipped
  + Model: One Equipment object in slot 1 with 5 subslots A-E with 4 containing Equipment (two expected only and two actual only)
* Slot 2
  + Expectation: Blocked by expectation in slot 1
  + Actual: Blocked by actual in slot 1
  + Model: No Equipment
* Slot 3
  + Expectation: Single slot card with subslots A-C where A and B are expected to be equipped
  + Actual: Single slot card with subslots A-C where A is equipped matching expectation, B is not equipped (not matching expectation) and C is unexpectedly equipped
  + Model: One Equipment object in slot 3 with 3 subslots with each containing an Equipment
* Slot 4
  + Expectation: Double slot card with subslots A-C where A and B are expected to be equipped
  + Actual: Single slot card of different type with subslots A-E where A is equipped with the wrong card type, B is not equipped C is not equipped as expected etc
  + Model: One Equipment object in slot with 5 subslots with 3 containing Equipments
* Slot 5
  + Expectation: Blocked by expectation in slot 4
  + Actual: Double slot card with subslots A & B where A is actually equipped
  + Model: One Equipment object in slot with 2 subslots with 1 containing an Equipment
* etc
* Slot 9 has a CIRCUIT\_PACK that matches the expectation (hence green actual column). It also has:
  + An SFP in slot A that matches expectation
  + Unexpectedly no SFP in slot B hence a mismatch
  + Unexpectedly an SFP in slot C hence the mismatch

Further explanation to be added in a later release.

The figure requires further development.

**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 0.0.11 1 June 2018 <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/]

[if cl.ownedComment->notEmpty()]<drop/>

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

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

[/for]<drop/>

[else]To be provided

[/if]<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. Often the word “physical” is used in the context of non-mechanical functional things. All non-mechanical functional things are considered under the functional model here regardless of how closely they are bound to the physical entities. All non-mechanical functional representations are considered as emergent abstractions and as virtualized (to some degree within a physical boundary). Mechanical functional things, such as fans, are not modelled in detail. [↑](#footnote-ref-1)
2. This model does not cover full inside/outside plant modelling. [↑](#footnote-ref-2)
3. The duct is not modelled as outside plant is beyond the scope of the model currently. [↑](#footnote-ref-3)
4. Clearly, the CD may have some other purpose and hence the association is multiplicity “\*”. [↑](#footnote-ref-4)
5. This association was not navigable from the LTP in 1.3.1 but is now navigable. [↑](#footnote-ref-5)
6. The protection model has only had very limited development so far and the model is clumsy in this area. [↑](#footnote-ref-6)
7. In a client-provider context Expectation is the client view of an agreement that the provider intends to satisfy. As it is not possible to remotely install a physical thing there is a decoupling between the request and the actual realization. This is very similar to the decoupling at a contractual interface where there is an agreement to provide something and hence an intention by the provider and an expectation from the client. The agreement is satisfied once the provider provides what was agreed. In the general case any properties could be constraints. In the case of the Equipment the agreement is more precise providing no choice of positioning. It is possible under some circumstances that position flexibility may also be allowed. This aspect is for further study. [↑](#footnote-ref-7)