<config>  
<output path=’C:\Users\ndavis\git\OnfInfoModelOutput\ModelDescriptions\TR-512.17\_OnfCoreIm-FoundationState.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-512.17

Foundation

(State)

Version 1.6

January 2024

ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.6

**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)

©2024 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)

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. |
| 1.5 | September 2021 | State aspects of TR-512.3 split out to form this document, TR-512.17. |
| 1.6 | January 2024 | Updated release and dates. |

# 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 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](../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 states.

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)).

# 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. This document addresses the states.

## 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 «Mature»/«Preliminary».

[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/>

The states described above apply to individual abstractions, it is also important to understand how the states of an abstraction are related to the states of the supporting abstractions. The relationship between the states of the abstractions is illustrated in Figure 3-2 below.



Figure - Relationship between abstractions

A description of the client abstraction, dependent abstraction and supporting abstraction is provided in section 3.1.6.

### Classes and attributes

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

[if (cl.qualifiedName.contains(‘StateModel::ObjectClasses’))]<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 the same context

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

If the assignmentState is POTENTIAL\_BUSY, POTENTIAL\_AVAILABLE or UNAVAILABLE the administrativeState should be LOCKED.

If the administrativeState is SHUTTING\_DOWN the assignmentState should be PENDING\_WITHDRAWAL.

In all other circumstances the states are independent.

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

The tables below list the states in the server context (supporting abstraction) that influence the states in the client context (dependent abstraction) in the same controller.

**Administrative state**

|  |  |  |
| --- | --- | --- |
| **Supporting abstraction administrativeState** | **Dependent abstraction administrativeState** | **Notes** |
| LOCKED | LOCKED | Service is blocked by the supporting abstraction |
| UNLOCKED | UNLOCKED |  |
| LOCKED | *Local modification*: Service is not blocked in the supporting abstraction |

**Operational state**

|  |  |
| --- | --- |
| **Supporting abstraction operationalState** | **Dependent abstraction operationalState** |
| ENABLED | ENABLED |
| DISABLED | DISABLED |

**Assignment state**

|  |  |  |
| --- | --- | --- |
| **Supporting abstraction assignmentState** | **Permitted dependent abstraction assignmentState** | **Notes** |
| **PLANNED** | PLANNED |  |
| **INSTALLED** | PLANNED | *Local modification:* The administrator may choose to delay showing the change to INSTALLED |
| INSTALLED |  |
| POTENTIAL\_AVAILABLE; | *Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource |
| POTENTIAL\_BUSY | *Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource |
| PENDING\_WITHDRAWAL\_FREE | *Local modification: Controlled by client using the resource*: Resource may be removed from this client. No impact on the supporting abstraction |
| SCHEDULED\_WITHDRAWAL | *Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource |
| SCHEDULED\_CAPACITY\_CHANGE | *Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource |
| **POTENTIAL\_AVAILABLE** | POTENTIAL\_AVAILABLE |  |
| PENDING\_WITHDRAWAL\_FREE | *Local modification*: *Controlled by client using the resource*: Resource may be removed from this client. No impact on the supporting entity |
| **POTENTIAL\_BUSY** | POTENTIAL\_BUSY; |  |
| PENDING\_WITHDRAWAL\_FREE | *Local modification*: *Controlled by client using the resource*: Resource may be removed from this client. |
| **PENDING\_WITHDRAWAL** | PENDING\_WITHDRAWAL |  |
| PENDING\_WITHDRAWAL\_FREE | *Local modification*: *Controlled by client using the resource*: The client is no longer using the resource |
| **UNAVAILABLE** | UNAVAILABLE |  |

No other states in the dependent abstraction (in the client context) have a dependency on the state of the supporting abstraction (in the server context).

The states in the client abstraction (in the server context of the “n+1” controller – as shown in Figure 3-2) should track the states of the dependent abstraction (in the client context of the “n” controller – as shown in Figure 3-2) except for PENDING\_WITHDRAWAL\_FREE which is controlled by the client using the resource.

The AdministrativeState in the server context is not visible in the client context. The client context may maintain an independent AdministrativeState.

The provider controls the assignmentState of the dependent abstraction that is visible to the client context as described in the table above.

### State transition diagrams

These state transition diagrams are preliminary sketches that may be refined in following releases.

#### Administrative State

Diagram

Description automatically generated

Figure - Administrative State

The shutting down state is used in a supporting abstraction when a dependent abstraction is in the assignment state of Installed (i.e., a client may be using the resource). The ShuttingDown state is not visible to the dependent abstraction. The assignmentState of the supporting abstraction should be changed to PENDING\_WITHDRAWAL when the AdministrativeState transitions to SHUTTING\_DOWN. After the assignmentState of the dependent abstraction has transitioned to PENDING\_WITHDRAWAL\_FREE the AdministrativeState of the supporting abstraction can transition to LOCKED and the assignmentState of the dependent abstraction should be changed to UNAVAILABLE or the abstraction should be deleted.

#### Operational State

Diagram

Description automatically generated

Figure - Operational State

The operationalState is controlled by the supporting hardware or software and is read only. The operationalState of a dependent abstraction in the INSTALLED state must match the operationalState of the supporting abstraction.

#### Assignment State

****

Figure - Assignment State

The allocation of a resource may be migrated between the use cases as described below.

From PermananetDedicatedResource to FixedCapacitySharedResource (1)

* From INSTALLED to SCHEDULED\_WITHDRAWAL

From FixedCapacitySharedResource to PermananetDedicatedResource (2)

* From SCHEDULED\_WITHDRAWAL or POTENTIAL\_AVAILABLE to INSTALLED

From VariableCapicitySharedResource to PermananetDedicatedResource (3)

* From SCHEDULED\_CAPACITY\_CHANGE to INSTALLED

From PermananetDedicatedResource to VariableCapicitySharedResource (4)

* From INSTALLED to SCHEDULED\_CAPACITY\_CHANGE

FixedCapacitySharedResource to VariableCapicitySharedResource (5)

* From SCHEDULED\_WITHDRAWAL to SCHEDULED\_CAPACITY\_CHANGE

From VariableCapicitySharedResource to FixedCapacitySharedResource (6)

* From SCHEDULED\_CAPACITY\_CHANGE to SCHEDULED\_WITHDRAWAL

If a resource is shared by two or more clients (dependent abstractions) when the supporting abstraction changes to INSTALLED, the dependent abstraction transitions to POTENTIAL\_AVAILABLE, POTENTIAL\_BUSY, SCHEDULED\_WITHDRAWAL or SCHEDULED\_CAPACITY\_CHANGE. A resource may be POTENTIAL\_AVAILABLE in all of the dependent abstractions or PENDING\_WITHDRAWAL in one dependent abstraction and POTENTIAL\_BUSY in all other dependent abstractions.

A resource in a supporting abstraction is moved to PENDING\_WITHDRAWAL when the provider of that resource intends to withdraw the resource from the client. The provider should indicate the time when the resource will be withdrawn and indicate if the withdrawal will be permanent or temporary (e.g., to allow network maintenance).

A resource is moved to PENDING\_WITHDRAWAL\_FREE when the user of that resource is no longer using the resource. This causes the corresponding resource in the client context (the dependent abstraction) to transition to PENDING\_WITHDRAWAL\_FREE. The provider may withdraw the resource after the specified time even if the client has not transitioned to PENDING\_WITHDRAWAL\_FREE.

When the resource in the dependent abstraction moves to PENDING\_WITHDRAWAL\_FREE the administrativeState of the supporting abstraction may be changed to LOCKED. If the withdrawal is temporary, the dependent abstraction should be changed to UNAVAILABLE. If the withdrawal is permanent the abstraction should be deleted.

Note that an abstraction in a server context my play the role of both the client abstraction (when viewed from the perspective of controller n in Figure 3-2) and the supporting abstraction (when viewed from the perspective of controller n+1 in Figure 3-2). In this case the POTENTIAL\_AVAILABLE and POTENTIAL\_BUSY states are used in the supporting abstraction.

### Use of states

#### Model context

Figure 3-6 below is an informal sketch of the relationship between the “platform,” that supports the functions, the logical resource model, that provides a view of the logical resources and the equipment and software models, that provide a view of the implementation platform. Note that this figure shows a highly simplified view of the equipment model and software model, only the major relationships of interest are shown. The client context and server context may be represented by constrain domains. The equipment model is in TR-512.6 and the software model is in TR-512.12.



Figure - Relationship between entities and abstractions

Figure 3-8 illustrates the relationship between the logical resource model and the supporting hardware or software platform, it does not show the full hardware or software model. In the equipment model and software model a field replaceable unit (FRU) or a running software module is represented by an object. These objects support one or more of the artefacts in the logical resource model. In the logical resource model, the lowest level objects are the aggregation of one or more of the artefacts.

*Example:*

* *An interface card on an Ethernet switch (a single FRU) may support 10 \* 1GE interfaces and thus 10 “Ethernet” LTPs.*
* *The PHY of each of the Ethernet interfaces may be on a separate pluggable module (FRU) so each Ethernet LTP is supported by both the interface card (FRU) and the pluggable module (FRU).*

The objects in a server context are referred to as the “supporting abstraction” and the objects in the client context are referred to as the “dependent abstraction”. The states of an object in a client context (the dependent abstraction) should be consistent with the state of the corresponding supporting object(s) in the server context (in the same controller). Also, the state of the object in the server context in the n+1 controller (client abstraction) should track the state of the object in the corresponding client context (in the n controller). Note that when the state of one or more of the supporting abstractions changes the state of the dependent abstraction and client abstraction will not be consistent until that change has been processed. In a recursive hierarchy, the client abstraction will play the role of supporting abstraction in the next level of the hierarchy.

Within a controller, the objects “directly” representing the implementation (e.g., in the server context in Controller n in Figure 3-6) should support the Administrative State. When the administrativeState is set to LOCKED, the implementation prevents the resource from being used. Abstraction in any other context (e.g., in the client context in controller n or server context in controller n+1) support of the administrative state is optional. Setting the administrativeState to LOCKED does not have any influence on the resource and it will continue to function normally.

The resources in a server context (i.e., the supporting abstractions) may be aggregated into a single object in a client context that is “mirrored” in the server context of the client controller by the client abstraction.

##### Alarms, fault isolation and control

The “lowest level” objects in the resource model are “directly connected” to the implementation and have access to the forwarding namespace. They can configure the resources that implement the logical functions and receive alarm/status information directly from the implementation. Fault location, to the level required for FRU replacement, can only be performed with information from this lowest level which provides access to the resource and physical name spaces.

When an object is presented in a client context, the alarm/status information is abstracted and aggregated, the relationship to the implementation is not visible from the client context.

#### Instance relationships

As described above, the supporting entity may be a “platform” or a managed object. The dependent entity is always a managed object. The relationships between the supporting and dependent abstractions may be:

|  |  |
| --- | --- |
| **Number of supporting abstractions/resources** | **Number of  dependent abstractions** |
| 1 | 1 |
| 1 | n |
| m | 1 |
| m | n |

The state of a dependent abstraction must be consistent with the state of the supporting abstraction(s). For the m:1 and m:n cases the states are considered in the order of the precedence described in section 3.2.6.2 below.

The state of the client abstraction (in the server context of controller n+1) should match the state of the dependent abstraction (in the client context of controller n).

##### Supporting:Dependent 1:1 Case and 1:n Case

The state of the dependent objects is set based on the state of the supporting abstraction and in some cases may be modified by the local controller as described in 3.1.4. For the 1:n case local modifications are made independently on each of the dependent abstractions.

##### Supporting:Dependent m:1 Case

The state of the dependent object is set based on the state of the *composite* supporting instance as described below. The precedence of the states in the tables below are used to determine the state of the *composite* supporting abstraction.

**Administrative state**

|  |  |
| --- | --- |
| **administrativeState** | **Precedence** |
| LOCKED | Highest |
| SHUTTING\_DOWN |  |
| UNLOCKED | Lowest |

**Operational state**

|  |  |
| --- | --- |
| **operationalState** | **Precedence** |
| DISABLED | Highest |
| ENABLED | Lowest |

**Assignment state**

| **assignmentState** | **Precedence** |
| --- | --- |
| PLANNED | Highest |
| UNAVAILABLE |  |
| PENDING\_WITHDRAWAL PENDING\_WITHDRAWAL\_FREE |  |
| POTENTIAL\_BUSY |  |
| POTENTIAL\_AVAILABLE |  |
| INSTALLED SCHEDULED\_WITHDRAWAL SCHEDULED\_CAPACITY\_CHANGE | Lowest |

The m:1 case has two sub-cases:

###### Simple

This arrangement of supporting abstraction and composite abstraction is illustrated in Figure 3-7 below.

Components

Composite

Figure - Component composite relationship

The state of the composite abstraction is determined by making a list of the states of the component supporting abstractions and arranging that list of states from the lowest to highest precedence. The resource policy defines the number of component supporting abstractions that must be in a given state (or in a lower precedence state) for the composite abstraction to be in that state.

Example of the application of resource policy

Using the assignment state enumerations as an example, then in the case of six component supporting abstractions with the following states:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Abstraction** | A | B | C | D | E | F |
| **State** | PLANNED | INSTALLED | POTENTIAL\_AVAILABLE | INSTALLED | POTENTIAL\_BUSY | POTENTIAL\_AVAILABLE |

Rearranging into an ordered list:

|  |  |  |
| --- | --- | --- |
|  | **Abstraction** | **State** |
| 1 | B | INSTALLED |
| 2 | D | INSTALLED |
| 3 | C | POTENTIAL\_AVAILABLE |
| 4 | F | POTENTIAL\_AVAILABLE |
| 5 | E | POTENTIAL\_BUSY |
| 6 | A | PLANNED |

The state of the composite abstraction is determined by applying the resource policy for the number of component supporting abstractions that must be in a given state (or in a lower precedence state) for the composite abstraction to be in that state. If the policy only requires 1 the composite state will be INSTALLED: If the policy requires 3 the composite state will be POTENTIAL\_AVAILABLE; If the policy requires 6 the composite state will be PLANNED.

###### Compound

This may be modelled by concatenating the appropriate set of simple cases described above. The state of the intermediate composite is evaluated for each (simple) group and is used to define the state of the intermediate component for the next stage of evaluation. That is, each of sets of “lower” components are evaluated, using the rules described above, to determine the intermediate composite states. These intermediate composite states are then used as the intermediate component state when evaluating the state of the next level of composite abstractions. Figure 3-8 below shows an example where three levels of evaluation used to map from the component states into the composite state. The sets of component abstractions that are grouped to form an intermediate composite abstraction is determined by the topological relationship between those abstractions. The compound case could, for example, be used to derive the state of a forwarding domain that is supported by a set of forwarding domains and links (each with its own intermediate composite state).



Figure - Compound component composite relationships

Some (or all) of the intermediate composite abstractions may be exposed to a client. Also, alternate intermediate aggregates of the same component resources may be exposed to a client as illustrated in Figure 3-9 below.



Figure - Alternate intermediate aggregates

In this case it is important that the state of the composite derived via intermediate aggregation 1 is the same as the state derived via intermediate aggregation 2.

##### Supporting:Dependent m:n case

Note: This does not represent a m:n protection scheme.

First the state of the m component abstractions are mapped into a single composite state as described above in 3.1.6.2.2. Then this composite state is used to derive the state of the dependent abstraction as described described above in 3.1.6.2.1. This would be used for example when a forwarding domain that is supported by forwarding domains and links is shared between n clients.

#### Protected entities

The state of an abstraction that is representing a protected resource is determined by the C&SC that is managing/representing the protection scheme. Note that a client (controller) may have a view of both the protected resource and the (unprotected) resources that support it.

#### Split entities

The view (abstractions) presented to a client controller cannot provide a more detailed view than that offered by the lowest level instances that have been created. When a client (controller) view of an entity is “split” as shown in the example in Figure 3-10 below where the original client FD is split into two FDs (X and Y). The links 2, 3 and LTPs 4, 5, 6, 7 are now exposed to the client. The client must retain the context of the original (parent) FD to understand the link between Client FD X and Client FD Y.



Figure - Split entity example

The states of the dependent resources provided by the server (controller) must reflect the states of the supporting resources as described above.

The new client FDs should be provided with new identifiers so that the identifier for the parent FD can be retained.

#### Merged entities

The state of the merged entity must be consistent with the state of the supporting resources as described above.

Considering the example above where the server controller is required to “merge” client FD X and client FD Y into a single FD. We have two cases:

* The parent FD is already visible (with an identifier) in the client context, in this case no further action is required.
* The parent FD is not visible in the client context. In this case a new ID for the parent FD should be created. This option must be used if the client has the option of viewing either level of abstraction (i.e., FD X, Y and Parent FD). The ID of either FD X or FD Y could be used for the parent FD. However, this is not recommended since it precludes the possibility of leaving FD X and FD Y visible to the client.

Appendix 1  
Examples of the use of temporal expressions for the assignmentState

**Example temporal expression for a fixed capacity resource shared by client A and B**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Period 1** | **Period 2** | **Period 3** |
| Server view | Assigned to A | Assigned to B | Free |
| Client A view | SCHEDULED\_WITHDRAWAL | POTENTIAL\_BUSY | POTENTIAL\_AVAILABLE |
| Client B view | POTENTIAL\_BUSY | SCHEDULED\_WITHDRAWAL | POTENTIAL\_BUSY\* |

**\*** The server may choose not to expose POTENTIAL\_AVAILABLE capacity to client B

**Example temporal expression for a variable capacity shared resource shared by clients A and B**

**Packet variable capacity shared resource allocation:**

Total capacity = 100:   
S allocated CIR £ 100:   
S allocated PIR £ 200: PIR allocated to one client £ 100   
Note: in this example the PIR is over-subscribed. Depending on the traffic loading a client may not be able to use the full allocated PIR.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Period 1** | **Period 2** | **Period 3** | **Period 4** |
| Capacity allocation | Table 1 | Table 2 | Table 3 | Table 4 |

|  |  |  |
| --- | --- | --- |
| **Table 1** | | |
|  | **CIR** | **PIR** |
| Client A | 50 | 80 |
| Client B | 20 | 30 |
| Unallocated | 30 | 90 |

|  |  |  |
| --- | --- | --- |
| **Table 2** | | |
|  | **CIR** | **PIR** |
| Client A | 0 | 0 |
| Client B | 90 | 100 |
| Unallocated | 10 | 100 |

|  |  |  |
| --- | --- | --- |
| **Table 3** | | |
|  | **CIR** | **PIR** |
| Client A | 0 | 10 |
| Client B | 20 | 30 |
| Unallocated | 80 | 160 |

|  |  |  |
| --- | --- | --- |
| **Table 4** | | |
|  | **CIR** | **PIR** |
| Client A | 10 | 20 |
| Client B | 90 | 100 |
| Unallocated | 0 | 80 |

**TDM variable capacity shared resource**

Total capacity = 100:   
S allocated capacity £ 100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Period 1** | **Period 2** | **Period 3** | **Period 4** |
| Capacity allocation | Table 1 | Table 2 | Table 3 | Table 4 |

|  |  |  |
| --- | --- | --- |
| **Table 1** | | |
|  | Allocated | Potential available[[1]](#footnote-1) |
| Client A | 50 | 30 |
| Client B | 20 | 30 |
| Unallocated | 30 | - |

|  |  |  |
| --- | --- | --- |
| **Table 2** | | |
|  | Allocated | Potential available 1 |
| Client A | 0 | 0 |
| Client B | 90 | 10 |
| Unallocated | 10 | - |

|  |  |  |
| --- | --- | --- |
| **Table 3** | | |
|  | Allocated | Potential available1 |
| Client A | 0 | 30 |
| Client B | 20 | 50 |
| Unallocated | 80 | - |

|  |  |  |
| --- | --- | --- |
| **Table 4** | | |
|  | Allocated | Potential available1 |
| Client A | 10 | 0 |
| Client B | 90 | 0 |
| Unallocated | 0 | - |

**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. The server may decide how much of the POTENTIAL\_AVAILABLE capacity is exposed to each of the clients [↑](#footnote-ref-1)