TM Forum Technical Report

Set Operators

TR292F

Maturity Level: **General availability (GA)**	Team Approved Date: 04-Jul-2024
Release Status: Production	Approval Status: TM Forum Approved
Version 3.6.0	IPR Mode: RAND

Notice

Table of Contents

Executive Summary

This document defines an ontology model of set operators.

Introduction

Requirement specification in intents often requires formulating conditions based on sets of things. For example, the state of a network resource shall be one of a set of "good" states in order for the system to meet the requirement. Another example is the requirement that a resource must not be allocated within certain distinct regions or sites. All this involves sets and logic regarding intersection of sets and membership of individuals within a set.

This document specifies the set operators model to introduce vocabulary for addressing these use cases.

Revision Information

This revision v3.6.0 of the set operators model is part of the TM Forum Intent Ontology (TIO) v3.6.0.

The revision v3.6.0 of this document replaces v.3.5.0 with the following changes:

- Minor editorial corrections.

1. Notation and Namespaces

The set operators ontology model depends on the following models:

Model	Prefix	Namespace	Published by	Purpose in the model
Set Operators Ontology	set	`http://tio.models.tmforum.org/tio/v3.6.0/SetOperators/`	TM Forum	(this model) Specification of set operators
Function Definition Ontology	fun	`http://tio.models.tmforum.org/tio/v3.6.0/FunctionOntology/`	TM Forum	Basic expression of functions
RDF version 1.1	`rdf`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	W3C	Providing fundamental modeling artifacts
RDF Schema 1.1	`rdfs`	`http://www.w3.org/2000/01/rdf-schema#`	W3C	Providing fundamental modeling artifacts
XML Schema	`xsd`	`http://www.w3.org/2001/XMLSchema#`	W3C	Providing of data types for literal objects
Time Ontology in OWL	t	`http://www.w3.org/2006/time#`	W3C	Expression of date and time [owltime]
Examples	ex:	`http://www..example.org/`	IANA	Reserved domain name for examples

Table 1.1: Model references

The set operators model is based on the Resource Description Framework (RDF) [rdf, rdf_mt, rdf_primer] and the Resource Description Framework Schema (RDFS) [rdfs] published by the World Wide Web Consortium (W3C).

The set operators model introduces functions that use containers as attributes or produce containers as result of their evaluation. Therefore, the function definition ontology is used.

2. Containers for modelling of sets

The TM Forum intent ontology uses the concepts of containers and collections from RDF to manage sets of things and express conditions based on elements of sets.

Collections in RDF have the structure of a linked list. It is built from `rdf:first` and `rdf:rest` properties. An instance of class `rdf:List` refers to the element of the collection using the `rdf:first` property, and `rdf:rest` refers to the list that contains the rest of the collection. Using `rdf:first` and `rdf:rest` statements repeatedly creates a graph that represents the collection with references to all its elements. The end of the list is represented by `rdf:nil`. In the TM Forum Intent ontology collections are primarily used to represent argument lists of functions as explained in the Function Definition Ontology.

RDFS defines containers as instances of class `rdfs:Container`. The classes `rdf:Bag`, `rdf:Seq` and `rdf:Alt` are subclasses of `rdfs:Container`. The member elements of a container would be specified using the `rdfs:member` property, or using container membership properties such as of `rdf:_1`, `rdf:_2`, `rdf:_3`, ... Containers of type `rdf:Seq` consider the order of its elements. Other types of containers do not preserve or imply element order.

The set operators model extends RDF and RDFS by introducing further properties for defining members of a container. It also introduces properties that represent logical operators. This allows the specification of conditions based on element memberships and container comparisons. These properties are functions according to the Function Definition Ontology defined within the TM Forum Intent Ontology.

3. Container Constructor Operators

The Set Operators Model specifies functions that represent containers based on other containers.

`set:union`

`<rdfs:Container> = set:union ( <rdfs:Container> ... )`The function `set:union` represents a container of all resources that are members of the containers specified in the function arguments.

For example:

```
ex:C1
  a rdfs:Container ;
  rdfs:member ex:RBS1 ;
  rdfs:member ex:RBS2
.
ex:C2
  a rdfs:Container ;
  rdfs:member ex:RBS3
.
ex:C3
  set:union ( ex:C1 ex:C2 ) ;
  rdfs:member ex:RBS4
.
```

This example defines two containers `ex:C1` and `ex:C2` explicitly by enumerating their members. The container ex:C3 is defined as containing members as a union of `ex:C1` and `ex:C2`. It therefore gained member resources `ex:RBS1`, `ex:RBS2` and `ex:RBS3` from the union. Another member `ex:RBS4` is added explicitly to `ex:C3`. As a result the container `ex:C3` has four members.

`set:intersection`

`<rdfs:Container> = set:intersection ( <rdfs:Container> ... )`

The function `set:intersection` represents a container of only those resources that are members of all containers specified in the arguments.

For example:

```
ex:C1
  a rdfs:Container ;
  rdfs:member ex:RBS1 ;
  rdfs:member ex:RBS2
.
ex:C2
  a rdfs:Container ;
  rdfs:member ex:RBS2 ;
  rdfs:member ex:RBS3
.
ex:C3
  a rdfs:Container ;
  rdfs:member ex:RBS1 ;
  rdfs:member ex:RBS2 ;
  rdfs:member ex:RBS4
.
ex:C4
  set:intersection ( ex:C1 ex:C2 ex:C3 ) ;
  rdfs:member ex:RBS4
.
```

This example defines the container ex:C4 as the intersection of three other containers. The only member all three have in common is ex:RBS2, thus this is the only member of the container ex:C4 from the intersection next to ex:RBS4, which is explicitly added.

`set:difference`

`<rdfs:Container> = set:difference ( <rdfs:Container> ... )`

The function `set:difference` represents a container of all members of the first argument container, which are not members of the other given containers as well.

For example:

```
ex:C1
  a rdfs:Container ;
  rdfs:member ex:RBS1 ;
  rdfs:member ex:RBS2
.
ex:C2
  a rdfs:Container ;
  rdfs:member ex:RBS2 ;
  rdfs:member ex:RBS3
.
ex:C3
  a rdfs:Container ;
  rdfs:member ex:RBS1 ;
  rdfs:member ex:RBS2 ;
  rdfs:member ex:RBS4
.
ex:C4
  set:difference ( ex:C3 ex:C2 ex:C1 ) ;
.
```

This example defines the container ex:C4 ad the difference between the container `ex:C3` and the two containers `ex:C1` and `ex:C2`. The only member of container ex:C3, which is neither a member of `ex:C1` nor `ex:C2` is `ex:RBS4`. This is therefore the only member of ex:C4.

4. Abstract Containers

RDF and RDFS provide the vocabulary for extensional definition of containers by listing all elements. In the formulation of requirements it would be useful to allow an intentional definition of containers. This refers to describing the properties of the container elements rather than enumerating them individually. An example would be the container representing a set of all users below the age of 20. Another example would be the container representing the set of all radio base station sites within a given geographical region. Instead of enumerating all elements of these containers, it would be possible to describe the properties of resources that are considered to be container members. These containers are referred to in the TM Forum Intent Ontology as "Abstract Containers".

Implementation note: Abstract containers to not imply that containers with all eligible elements must be explicitly created and maintained in the knowledge base by enumerating all defined elements that meet the element description. Instead, the underlying knowledge reasoning implementation can evaluate the member allocation against the knowledge base dynamically.

`set:resourcesOfType`

`<rdfs:Container> = set:resourcesOfType ( <rdfs:Class> ... )`

The function `set:resourcesOfType` represents a container of all resources that are instances of the given class or its subclasses. Closed world assumption applies.

For example:

```
ex:RBS1
  a ex:RadioBaseStation
.
ex:RBS2
  a ex:RadioBaseStation
.
ex:C1
  set:resourcesOfType ( ex:RadioBaseStation )
.
```

This example shows the definition of two instances of class `ex:RadioBaseStation`. The container `ex:C1` is then defined as having all resources in the knowledge base as members, which are instances of that class. Assuming that there are no additional instances than the two defined in the example, the container `ex:C1` is assumed to have members `ex:RBS1` and `ex:RBS2`.

The dynamic character of this container definition becomes clear if we consider that later another instance of that class is added to the knowledge base. As soon as this is done, the container gains another member implicitly. This means the member association is dynamically following the content of the knowledge base. Also, closed world assumption applies, and this means that the system is only considering what its own knowledge base contains. The possibility that there might be other domains with additional knowledge bases and that they might contain further instances of that class `ex:RadioBaseStation`, would not change the evaluation of the locally defined container `ex:C1`.

`set:typesOfMembers`

`<rdfs:Container> = set:typesOfMembers ( <rdfs:Container> ... )`

The function `set:typesOfMembers` represents a container of all classes the members of the argument containers are instances of.

For example:

```
ex:Node1
  a ex:RadioBaseStation
.
ex:Node2
  a ex:RadioBaseStation
.
ex:Node3
  a ex:Gateway
.
ex:C1
  a rdfs:Container ;
  rdfs:member ex:Node1 ;
  rdfs:member ex:Node2 ;
  rdfs:member ex:Node3
.
ex:C2
  a rdfs:Container ;
  rdfs:member ex:Node2
.
ex:Ctypes
  set:typesOfMembers ( ex:C1 ex:C2 )
.
```

This example defines three instances representing network nodes. Two of them are of type `ex:RadioBaseStation` and one of type `ex:Gateway`. Then the example defines two containers explicitly with sets of these nodes as members. Finally, the example defines the container `ex:Ctypes` with the function `set:typesOfMembers` defining its members. As a result the container `ex:Ctypes` is considered to have the two members `ex:RadioBaseStation` and `ex:Gateway`. Even though there are multiple members across the argument containers that have the type `ex:RadioBaseStation`, it appears only once in the container `ex:Ctypes`.

`set:resourcesWithProperty`

`<rdfs:Container> = set:resourcesWithProperty ( <rdf:Property> ... )`

The function `set:resourcesWithProperty` represents a container of all resource instances that are subject of any of the properties given as argument. Closed world assumption applies.

For example:

```
ex:Node1
  ex:temp 30
.
ex:Node2
.
ex:Node3
  ex:temperature 32
.
```

```
ex:Ctemp
  set:resourcesWithProperty ( ex:temp ex:temperature )
.
```

This example defines three instances representing nodes and some of them have temperature properties. Furthermore, there are two different properties that assign a temperature reading to the instance. The container `ex:Ctemp` is defined using the `set:resourcesWithProperty` function. Its arguments contain both variants of temperature properties. As a result, `ex:Node1` and `ex:Node3` are considered members of the collection `ex:Ctemp`.

set:resourcesWithPropertyObject

`<rdfs:Container> = set:resourcesWithPropertyObject ( <rdf:Property> <rdfs:Resource> ... )`

The function `set:resourcesWithPropertyObject` represents a container of all resources, to which the property specified in the first argument is used to assign any of the objects of the remaining arguments. Closed world assumption applies.

For Example

```
ex:Node1
  ex:region ex:SubnetA
.
ex:Node2
  ex:region ex:SubnetB ;
  ex:neighbor ex:SubnetC
.
ex:Node3
  ex:region ex:SubnetA ;
  ex:region ex:SubnetB
.
ex:C1
  set:resourcesWithPropertyObject ( ex:region ex:SubnetA ex:SubnetC )
.
```

This example defines three nodes participating in regions within the network. The container `ex:C1` is defined using the `set:resourcesWithPropertyObject` function. It specifies that members of the resulting container are resources to which `ex:SubnetA` or `ex:SubnetC` are assigned using the property `ex:region`. There are no instances to with `ex:SubnetC` is associated with the property `ex:region`. But two instances with `ex:SubnetA`. Therefore, `ex:Node1` and `ex:Node3` are considered to be members of `ex:C1`.

`icm:valuesOfObjectProperty`

`<rdfs:Container> = icm:valuesOfObjectProperty ( <rdf:Property> <rdfs:Resource> ... )`

The function `set:valuesOfObjectProperty` represents a container of all values that are assigned with the property given as first argument to the objects provided by the remaining arguments. If and object has multiple values assigned with this property, or if multiple objects are specified, the result is a superset of all values assigned to provided objects with this property.

For example:

```
ex:C1
  a log:Condition ;
  quan:smaller ( quan:meanOfSet ( [ icm:valuesOfObjectProperty ( ex:latency
ex:Slice1 ) ] )
                 "10ms"^^quan:quantity
               )
.
```

This example uses the function `icm:valuesOfObjectProperty` to represent a container of all latency values assigned to a slice. It then calculates the mean of these values and checks if it is smaller than 10ms. This expresses a condition.

5. Set Logic Operators

The functions defined in this chapter have a boolean truth value as result. They represent logical evaluations regarding the resources and containers in the function arguments.

`set:elementOf`

`<xsd:boolean> = set:elementOf ( <rdfs:Resource> <rdfs:Container> ... )`

The function `set:elementOf` represents the result of an evaluation. It assumes the boolean type result value "true", if the resource given in the first argument is a member of all containers given by the remaining arguments.

For example:

```
ex:RegionA
  a rdfs:Container ;
  rdfs:member ex:Node1 ;
  rdfs:member ex:node2
.
ex:Condition1
  a log:Condition ;
  set:elementOf ( ex:Node1 ex:RegionA )
.
```

This example defines the result of a condition through the function `set:elementOf`. It checks if the resource `ex:Node1` is member of the collection `ex:RegionA`. As this is currently the case, the condition assumes the boolean value "true".

`set:empty`

`<xsd:boolean> = set:empty ( <rdfs:Container> ... )`

The function `set:empty` represents the result of an evaluation. It assumes the boolean type result value "true", if none of the containers given as argument has any members.

For example:

```
ex:UserGroup1
  a rdfs:Container ;
  rdfs:member ex:User1 ;
  rdfs:member ex:User2
.
ex:UserGroup2
  a rdfs:Container
.
ex:Condition1
  a log:Condition ;
  set:empty ( ex:UserGroup1 ex:UserGroup2 )
.
```

This example defines the result of a condition through the function set:empty. It checks if the containers `ex:UserGroup1` and `ex:UserGroup2` are empty. Currently, `ex:UserGroup2` is empty, but `ex:UserGroup1` has two members. Therefore, the function and consequently the condition object `ex:Condition1` assume the value "false".

`set:includedIn`

`<xsd:boolean> = set:includedIn ( <rdfs:Container> <rdfs:Container> ... )`

The function `set:empty` represents the result of an evaluation. It assumes the boolean type result value "true", if all members of the container in the first argument are also members of each container given by the remaining arguments.

For example

```
ex:UserGroup1
  a rdfs:Container ;
  rdfs:member ex:User2, ex:User4
.
ex:UserGroup2
  a rdfs:Container ;
  rdfs:member ex:User1, ex:User2, ex:User3, ex:User4
.
ex:UserGroup3
  a rdfs:Container ;
  rdfs:member ex:User2, ex:User3, ex:User4
.
ex:Condition1
  a log:Condition ;
  set:includedIn ( ex:UserGroup1 ex:UserGroup2 ex:UserGroup3 )
.
```

This example defines the result of a condition through the function `set:includedIn`. It checks if all members of container `ex:UserGroup1` are also members of container `ex:UserGroup2` and `ex:UserGroup3`. The two members of `ex:UserGroup1` are `ex:User2` and `ex:User4`. Both of them are also members of `ex:UserGroup2` and `ex:UserGroup3`. Therefore, the function and consequently the condition object `ex:Condition1` assume the boolean value "true".

`set:intersectsWith`

```
<xsd:boolean> = set:intersectsWith ( <rdfs:Container> <rdfs:Container>
... )
```

The function `set:intersectsWith` represents the result of an evaluation. It assumes the boolean type result value "true", if there are member objects in the first argument container that are also members in all other containers given as arguments.

For example:

```
ex:UserGroup1
  a rdfs:Container ;
  rdfs:member ex:User2, ex:User3, ex:User4
.
ex:UserGroup2
  a rdfs:Container ;
  rdfs:member ex:User1, ex:User2, ex:User4
.
ex:UserGroup3
  a rdfs:Container ;
  rdfs:member ex:User3, ex:User5
.
ex:Condition1
  a log:Condition ;
  set:intersectsWith ( ex:UserGroup1 ex:UserGroup2 ex:UserGroup3 )
.
```

This example defines the result of a condition through the function `set:intersectsWith`. It checks if there are members in container `ex:UserGroup1` that are also members of the containers `ex:UserGroup2` and `ex:UserGroup3`. The resources `ex:User2` and `ex:User4` are members of `ex:UserGroup1` and `ex:UserGroup2`. The Containers `ex:UserGroup1` and `ex:UserGroup3` have the member `ex:User3` in common. As `ex:UserGroup1` has a member in common with both other user groups, the function result is "`true`". It does not matter that `ex:UserGroup2` and `ex:UserGroup3` do not have common members, because the function only checks for the first argument against all other arguments.

`set:identical`

`<xsd:boolean> = set:identical ( <rdfs:Container> ... )`

The function `set:identical` represents the result of an evaluation. It assumes the boolean type result value "`true`", all containers given as attributes have exactly the same set of members.

For example:

```
ex:UserGroup1
  rdfs:member ex:User1, ex:User2
.
ex:UserGroup2
  rdfs:member ex:User1, ex:User2
.
ex:UserGroup3
  rdfs:member ex:User1, ex:User2, ex:User3
.
ex:Condition1
  a log:Condition ;
  set:identical ( ex:UserGroup1 ex:UserGroup2 ex:UserGroup3 )
.
```

This example defines the result of a condition through the function `set:identical`. It checks if all containers given as argument have exactly the same members. In this example, the containers `ex:UserGroup1` and `ex:UserGroup2` are identical, but `ex:UserGroup3` has one more member. Therefore, the function `set:identical` and consequently the condition `ex:Condition1` assume the boolean result "`false`".

`set:isMember`

`<xsd:boolean> = set:isMember ( <rdf:Resource> <rdf:Container> ... )`

The function `set:isMember` has at least two arguments. It provides the result "true" if the resource provided as first argument is a member of any container provided by the remaining arguments.

For example:

```
ex:GoldUsers
  a rdfs:Container ;
  rdfs:member ex:User1, ex:User2, ex:User4
.
ex:IsUserGold
  a log:Condition ;
  set:isMember ( ex:User3 ex:GoldUsers )
.
```

This example specifies a condition that checks if user 3 is a gold user. The set of gold users is represented by the container `ex:GoldUsers`. The condition uses the function set:isMember to check if `ex:User1` is a member of this container.

`set:forAll`

```
<xsd:boolean> = set:forAll ( <rdf:Resource> <rdf:Container>
<rdf:Resource> ... )
```

The function `set:forAll` has at least three arguments. The first argument represents any individual member from the container provided in the second argument. The third and all following arguments are resources with Boolean value. This can for example be a function that evaluates to a Boolean result. The function `set:forAll` has a Boolean type result directly derived from the third argument. The `set:forAll` function replicates the evaluation of the third argument for every member of the second argument. This allows to express that a condition set by the third argument needs to be met by every member of the provided container.

The first argument is a resource use do represent individual members of the container. It can be used in the expression of the condition or evaluation specified by the third argument.

For example:

```
ex:T1
  a icm:Target ;
  rdfs:member ex:Slice1, ex:Slice2
.
ex:E2
  a icm:Expectation ;
  icm:target ex:T1 ;
  log:allOf ( ex:HighAvailability )
.
ex:HighAvailability
  a log:Condition ;
  set:forAll (
    _:X
    [ icm:valuesOfTargetProperty ( ex:availability ) ]
    [ quan:greater ( _:X "0.97"^^quan:Quantity ) ]
  )
.
```

This example defines a condition used within an expectation. The condition uses `set:forAll` to express that a condition shall be checked for all members of a container. The condition to be checked is a comparison between two quantities using the quan:greater function. The container providing individual members to be checked is provided by the second argument. In this example the function `icm:valuesOfTargetProperty` is used. It provides a container of values assigned to members of the target container, with the property `ex:availability`. The first argument of set:forAll represents individual values from this container. Here the blind-node _:X is used as representative of a member value. The same _:X is then also used as first argument of the quan:greater function. As a result, the `quan:greater` function is evaluated as many times as there are availability values in the container of the second argument. Each individual evaluation is done with a different input covering all values in the container. The function set:forAll evaluates to "true" if all individual evaluations of quan:greater provide "true" as result. This example effectively checks, if every target resource has an availability greater than 0.97.

6. Time based selection

Members of containers might be annotated with time stamps. Using the time-stamp information would allow selecting and filter members according to time-based conditions.

`set:membersBefore`

```
<rdfs:Container> = set:membersBefore ( <rdf:property> <t:Instant>
<rdfs:Container> ... )
```

The function `set:membersBefore` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time stamp is before the time stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

`set:membersAfter`

```
<rdfs:Container> = set:membersAfter ( <rdf:property> <t:Instant>
<rdfs:Container> ... )
```

The function `set:membersAfter` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time stamp is after the time stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

`set:membersSameTime`

```
<rdfs:Container> = set:membersSameTime ( <rdf:property> <t:Instant>
<rdfs:Container> ... )
```

The function `set:membersSameTime` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time-stamp shows the same time as the time-stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

`set:membersWhile`

```
<rdfs:Container> = set:membersSameTime ( <rdf:property> <t:Interval>
<rdfs:Container> ... )
```

The function `set:membersWhile` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the time-stamp to be used in the evaluation of this function. The second parameter specifies time interval in time expressed as individual of the class `t:Interval` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time-stamp lies within the time interval provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

`set:newestMember`

```
<rdf:Resource> = set:newestMembers ( <rdf:property> <rdfs:Container> ...
)
```

The function `set:newestMember` selects the newest member of the container according to a time-stamp property of the member object. The first parameter refers to the property of a member that assigns a time-stamp. This is the time-stamp to be used in the evaluation of this function. The third and all remaining parameters are containers.

The function provides the resource as result that is the newest member object within the union of all containers provided as parameters as indicated by the property of the member object that was specified as first function parameter.

`set:oldestMembers`

```
<rdf:Resource> = set:oldestMembers ( <rdf:property> <rdfs:Container> ...
)
```

The function `set:oldestMember` selects the oldest member of the container according to a time-stamp property of the member object. The first parameter refers to the property of a member that assigns a time-stamp. This is the time-stamp to be used in the evaluation of this function. The third and all remaining parameters are containers.

The function provides the resource as result that is the oldest member object within the union of all containers provided as parameters as indicated by the property of the member object that was specified as first function parameter.

7. Administrative Appendix

7.1. Document History

7.1.1. Version History

Version Number	Date Modified	Modified by:	Description of changes
1.0.0	07-Feb-2023	Alan Pope	Final edits prior to publication
3.0.0	11-Apr-2023	Alan Pope	Final edits prior to publication
3.2.0	15-Aug-2023	Alan Pope	Final edits prior to publication
3.4.0	29-Feb-2024	Alan Pope	Final edits prior to publication
3.5.0	03-May-2024	Alan Pope	Final edits prior to publication
3.6.0	04-Jul-2024	Alan Pope	Final edits prior to publication

7.1.2. Release History

Release Status	Date Modified	Modified by:	Description of changes
Pre-production	07-Feb-2023	Alan Pope	Initial release
Pre-production	17-Mar-2023	Adrienne Walcott	Updated to Member Evaluated status
Pre-production	11-Apr-2023	Alan Pope	Updated to v3.0.0
Pre-production	15-May-2023	Adrienne Walcott	Updated to Member Evaluated status
Pre-production	15-Aug-2023	Alan Pope	Updated to v3.2.0
Pre-production	18-Sep-2023	Adrienne Walcott	Updated to Member Evaluated status
Pre-production	29-Feb-2024	Alan Pope	Updated to v3.4.0
Production	26-Apr-2024	Adrienne Walcott	Updated to reflect TM Forum Approved status
Pre-production	03-May-2024	Alan Pope	Updated to v3.5.0
Production	28-Jun-2024	Adrienne Walcott	Updated to reflect TM Forum Approved status
Pre-production	04-Jul-2024	Alan Pope	Updated to v3.6.0
Production	30-Aug-2024	Adrienne Walcott	Updated to reflect TM Forum Approved status

7.2. Acknowledgments

Team Member (@mention)	Company	Role*
Jörg Niemöller	Ericsson	Author, Project Co-Chair
Kevin McDonnell	Huawei	Project Co-Chair
Yuval Stein	Amdocs	Project Co-Chair
Kamal Maghsoudlou	Ericsson	Key Contributor
Leonid Mokrushin	Ericsson	Key Contributor
Marin Orlić	Ercisson	Key Contributor
Aaron Boasman-Patel	TM Forum	Additional Input
Alan Pope	TM Forum	Additional Input
Dave Milham	TM Forum	Additional Input
Xiao Hongmei	Inspur	Reviewer

**Select from: Project Chair, Project Co-Chair, Author, Editor, Key Contributor, Additional Input, Reviewer*

8. Appendix A: Vocabulary Reference

This Appendix chapter contains a reference definition of all model vocabulary. It is sorted alphabetically:

8.1. difference

The function `set:difference` represents a container of all members of the first argument container, which are not members of the other given containers as well. If no argument is given, the resulting container is empty. If only one argument is given, the resulting container is identical to the argument.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: None or any number of arguments
Argument types: All arguments are of type `rdfs:Container`

8.2. elementOf

The function `set:elementOf` represents the result of an evaluation. It assumes the boolean type result value "true", if the resource given in the first argument is a member of all the containers given by the remaining arguments.

Instance of: `fun:Function`Result type: `xsd:boolean`Arity: At least two
Argument types: The first argument is of type `rdfs:Resource` and all remaining arguments are of type `rdfs:Container`

8.3. empty

The function `set:empty` represents the result of an evaluation. It assumes the boolean type result value "true", if none of the containers given as arguments has any members.

Instance of: `fun:Function`Result type: `xsd:boolean`Arity: At least one
Argument types: All arguments are of type `rdfs:Container`

8.4. forAll

The function `set:forAll` has at least three arguments. The first argument represents any individual member from the container provided in the second argument. The third and all following arguments are resources with Boolean value. This can for example be a function that evaluates to a Boolean result. The function `set:forAll` has a Boolean type result directly derived from the third argument. The `set:forAll` function replicates the evaluation of the third argument for every member of the second argument. This allows to express that a condition set by the third argument needs to be met by every member of the provided container. The first argument is a resource use do represent individual members of the container. It can be used in the expression of the condition or evaluation specified by the third argument.

Instance of: `fun:Function`Result type: `xsd:boolean`Arity: At least three
Argument types: The first argument is of type rdf:Resource, the second argument is of type `rdfs:Container`, the third and all further arguments are of type `rdf:Resource`.

8.5. includedIn

The function `set:includedIn` represents the result of an evaluation. It assumes the boolean type result value "true", if all members of the container in the first argument are also members of each container given by the remaining arguments.

Instance of: `fun:Function`Result type: `xsd:boolean`Arity: At least two
Argument types: All arguments are of type `rdfs:Container`

8.6. intersection

The function `set:intersection` represents a container of only those resources that are members of all containers specified in the arguments. If no arguments are given, the resulting container is empty. If only one argument is given, the resulting container is identical to the argument.

Instance of: `fun:Function`
Result type: `rdfs:Container`
Arity: None or any number of arguments
Argument types: all arguments are of type `rdfs:Container`

8.7. intersectsWith

The function `set:intersectsWith` represents the result of an evaluation. It assumes the boolean type result value "true", if there are member objects in the first argument container that are also members in all other containers given as arguments.

Instance of: `fun:Function`
Result type: `xsd:boolean`
Arity: at least two
Argument types: all arguments are of type `rdfs:Container`

8.8. isMember

The function `set:isMember` has at least two arguments. It provides the result "true" if the resource provided as first argument is a member of any container provided by the remaining arguments.

Instance of: `fun:Function`
Result type: `xsd:boolean`
Arity: at least two
Argument types: The first argument is of type rdf:Resource, all further arguments are of type `rdfs:Container`

8.9. membersAfter

The function `set:membersAfter` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time stamp is after the time stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: At least three
Argument types: The first Argument is of type `rdf:Property`, The second argument is of type `t:Instance` and all further arguments are of type `rdfs:Container`

8.10. membersBefore

The function `set:membersBefore` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time stamp is before the time stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: At least three
Argument types: The first Argument is of type `rdf:Property`, The second argument is of type `t:Instance` and all further arguments are of type `rdfs:Container`

8.11. membersSameTime

The function `set:membersSameTime` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the timestamp to be used in the evaluation of this function. The second parameter specifies a point in time expressed as individual of the class `t:Instant` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time-stamp shows the same time as the time-stamp provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: At least three
Argument types: The first Argument is of type `rdf:Property`, The second argument is of type `t:Instance` and all further arguments are of type `rdfs:Container`

8.12. membersWhile

The function `set:membersWhile` represents the result of a time based container member filtering. The first parameter refers to the property of a member that assigns a time-stamp to a member object. This is the time-stamp to be used in the evaluation of this function. The second parameter specifies time interval in time expressed as individual of the class `t:Interval` defined in the time ontology in OWL [owltime]. The third and all remaining parameters are containers.

The function provides a container as result. The members of this result container are members of the union of all containers provided as parameters, which meet the condition that their time-stamp lies within the time interval provided as second parameter. The property provided as first parameter is the one used to evaluate the condition. This property needs to be used for a member of the input containers for the member object to be considered in the evaluation and included in the result.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: At least three
Argument types: The first Argument is of type `rdf:Property`, The second argument is of type `t:Interval` and all further arguments are of type `rdfs:Container`

8.13. newestMember

The function `set:newestMember` selects the newest member of the container according to a time-stamp property of the member object. The first parameter refers to the property of a member that assigns a time-stamp. This is the time-stamp to be used in the evaluation of this function. The third and all remaining parameters are containers. The function provides the resource as result that is the newest member object within the union of all containers provided as parameters as indicated by the property of the member object that was specified as first function parameter.

Instance of: `fun:Function`Result type: `rdf:Resource`Arity: At least two
Argument types: The first Argument is of type `rdf:Property`, all further arguments are of type `rdfs:Container`

8.14. oldestMember

The function `set:oldestMember` selects the oldest member of the container according to a time-stamp property of the member object. The first parameter refers to the property of a member that assigns a time-stamp. This is the time-stamp to be used in the evaluation of this function. The third and all remaining parameters are containers. The function provides the resource as result that is the oldest member object within the union of all containers provided as parameters as indicated by the property of the member object that was specified as first function parameter.

Instance of: `fun:Function`Result type: `rdf:Resource`Arity: At least two
Argument types: The first Argument is of type `rdf:Property`, all further arguments are of type `rdfs:Container`

8.15. resourcesOfType

The function `set:resourcesOfType` represents a container of all resources that are instances of the given classes or its subclasses. Closed world assumption applies. If no argument is given, the resulting container is empty.

Instance of: `fun:Function`
Result type: `rdfs:Container`
Arity: None or any number of arguments
Argument types: all arguments are of type `rdfs:Class`

8.16. resourcesWithProperty

The function `set:resourcesWithProperty` represents a container of all resource instances that are subject of the properties given as argument. Closed world assumption applies. If no arguments are given, the resulting container is empty.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: None or any number of arguments
Argument types: all arguments are of type `rdfs:Property`

8.17. resourcesWithPropertyObject

The function `set:resourcesWithPropertyObject` represents a container of all resources, to which the property specified in the first argument is used to assign any of the objects of the remaining arguments. Closed world assumption applies.

Instance of: `fun:Function`Result type: `rdfs:Container`Arity: at least two
Argument types: all arguments are of type `rdfs:Property` and all remaining arguments are of type `rdfs:Resource`

8.18. typesOfMembers

The function `set:typesOfMembers` represents a container of all classes the elements of the containers in the function arguments are instances of. Duplicates are removed from the resulting container and only included once as member. If no arguments are given, the resulting container is empty.

Instance of: `fun:Function`
Result type: `rdfs:Container`
Arity: None or any number of arguments
Argument types: all arguments are of type `rdfs:Container`

8.19. union

The function `set:union` represents a container of all resources that are members of the containers specified in the function arguments. If no arguments are given, the resulting container is empty.

Instance of: `fun:Function`
Result type: `rdfs:Container`
Arity: None or any number of arguments
Argument types: all arguments are of type `rdfs:Container`

8.20. valuesOfObjectProperty

The function `set:valuesOfObjectProperty` represents a container of all values that are assigned with the property given as first argument to the objects provided by the remaining arguments. If and object has multiple values assigned with this property, or if multiple objects are specified, the result is a superset of all values assigned to provided objects with this property.

```
<rdfs:Container> = icm:valuesOfObjectProperty ( <rdf:Property>
<rdfs:Resource> ... )
```

Instance of: `fun:Function`
Result type: `rdfs:Container`
Arity: At least 2 arguments
Argument types: The first argument and property of type `rdf:Property`. All remaining arguments are `rdfs:Resources`.

8.20.1. Vocabulary

The object set`:Vocabulary` is a container of all model elements.

Instance of: `rdfs:Container`