USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR
OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS


For a complete list of trademarks, see: [https://www.omg.org/legal/tm_list.htm](https://www.omg.org/legal/tm_list.htm). All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.
OMG’S ISSUE REPORTING PROCEDURE

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page https://www.omg.org, under Documents, Report a Bug/Issue.
# Table of Contents

0 Preface.........................................................................................................................5
1 Scope................................................................................................................................1
2 Conformance.....................................................................................................................3
3 Normative References.......................................................................................................5
4 Terms and Definitions.........................................................................................................7
5 Symbols...............................................................................................................................9
6 Introduction.......................................................................................................................11
   6.1 API and Services Architecture....................................................................................11
   6.2 Document Conventions...............................................................................................12
   6.3 Document Organization...............................................................................................13
   6.4 Acknowledgements.....................................................................................................13
7 Platform Independent Model (PIM)..................................................................................15
   7.1 API Model..................................................................................................................15
      7.1.1 Record................................................................................................................15
      7.1.2 Project Data Versioning.....................................................................................16
      7.1.3 ExternalData and ExternalRelationship.............................................................19
      7.1.4 Query..................................................................................................................20
   7.2 API Services..............................................................................................................21
      7.2.1 ProjectService....................................................................................................21
      7.2.2 ElementNavigationService...............................................................................22
      7.2.3 ProjectDataVersioningService..........................................................................23
      7.2.4 QueryService....................................................................................................26
      7.2.5 ExternalRelationshipService............................................................................27
      7.2.6 ProjectUsageService..........................................................................................27
8 Platform Specific Models (PSMs)....................................................................................29
   8.1 REST/HTTP PSM.......................................................................................................29
      8.1.1 Overview............................................................................................................29
      8.1.2 PIM API Model - REST/HTTP PSM Model Mapping..........................................29
      8.1.3 PIM API Services - REST/HTTP PSM Endpoints Mapping.................................30
   8.2 OSLC 3.0 PSM..........................................................................................................35
      8.2.1 Overview............................................................................................................35
      8.2.2 OSLC Nomenclature..........................................................................................36
      8.2.3 PIM API Model – OSLC PSM Resource Mapping..............................................37
      8.2.4 PIM API Services – OSLC PSM Service Mapping...............................................39
A Annex: Conformance Test Suite......................................................................................49
   A.1 ProjectService Conformance Test Cases.................................................................49
   A.2 ElementNavigationService Conformance Test Cases..............................................51
   A.3 ProjectDataVersioningService Conformance Test Cases........................................54
   A.4 QueryService Conformance Test Cases....................................................................64
   A.5 ExternalRelationshipService Test Cases.................................................................66
   A.6 ProjectUsageService Conformance Test Cases.......................................................68
   A.7 Cross-Cutting Conformance Test Cases....................................................................69
B Annex: API and Services Examples and Cookbook.........................................................75
   B.1 Examples..................................................................................................................75
   B.2 Cookbook..................................................................................................................86
List of Tables

1. Operations ...................................................................................................................................................................................... 21
2. Operations ...................................................................................................................................................................................... 22
3. Operations ...................................................................................................................................................................................... 23
4. Operations ...................................................................................................................................................................................... 23
5. Operations ...................................................................................................................................................................................... 23
6. Operations ...................................................................................................................................................................................... 26
7. Operations ...................................................................................................................................................................................... 27
8. Operations ...................................................................................................................................................................................... 27
9. PIM API Model - REST/HTTP PSM Model Mapping Table ........................................................................................................ 29
10. PIM to REST / HTTP PSM Mapping ........................................................................................................................................ 30
11. PIM Concept to OSLC Resource type Mapping ......................................................................................................................... 38
12. PIM API Services - OSLC Services Mapping ................................................................................................................................ 39
List of Figures

1. API and Services Provider and Consumer ......................................................................................................................... 3
2. API and Services Architecture ............................................................................................................................................. 11
3. Use of PIM and PSMs by Providers and Consumers ........................................................................................................... 12
4. Types of Records ........................................................................................................................................................................ 15
5. Project Data Versioning API Model ...................................................................................................................................... 16
6. External Relationship API Model .......................................................................................................................................... 19
7. Query API Model .......................................................................................................................................................................... 20
8. ProjectService Operations ...................................................................................................................................................... 21
10. ProjectDataVersioningService Operations ............................................................................................................................ 23
11. QueryService Operations ....................................................................................................................................................... 26
12. ExternalRelationshipService Operations ............................................................................................................................... 27
13. ProjectUsageService Operations ........................................................................................................................................... 27
0 Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG’s specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at https://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling, and vertical domain frameworks. All OMG Specifications are available from the OMG website at: https://www.omg.org/spec

All of OMG’s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
9C Medway Road, PMB 274
Milford, MA 01757
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320

Email: pubs@omg.org

Certain OMG specifications are also available as ISO standards. Please consult https://www.iso.org

Issues

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page https://www.omg.org, under Specifications, Report an Issue.
1 Scope

The purpose of this standard is to specify the Systems Modeling Application Programming Interface (API) and Services that provide standard services to access, navigate, and operate on KerML-based models [KerML], and, in particular, SysML models [SysML]. The standard services facilitate interoperability both across SysML modeling environments and between SysML modeling environments and other engineering tools and enterprise services.

The Systems Modeling API and Services specifies the types and details of the requests that can be made and responses that can be received by software applications that are consuming the services to software applications that are providing the services.

The Systems Modeling API and Services specification includes the Platform Independent Model (PIM) - see Clause 7 - and two Platform Specific Models (PSMs) - see Clause 8: REST/HTTP PSM and OSLC PSM.
2 Conformance

This specification defines the Systems Modeling API and Services that provide standard services to access, navigate, and operate on KerML-based models [KerML] and, in particular, SysML models [SysML]. The specification comprises this document together with the content of the machine-readable files listed on the cover page. If there are any conflicts between this document and the machine-readable files, the machine-readable files take precedence.

A **Systems Modeling API and Services Provider** is a software application that provides the services defined in this specification.

A **Systems Modeling API and Services Consumer** is a software application that consumes the services defined in this specification and provided by the Service Provider.

Consumers send requests to Providers and receive responses with results, as illustrated in **Fig. 1** below.

For brevity, this specification uses the phrase **Service Provider** for **Systems Modeling API and Services Provider**, and the term **Service Consumer** for **Systems Modeling API and Services Consumer**.

![Figure 1. API and Services Provider and Consumer](image)

A Service Provider can conform to this specification at the PSM or PIM level.

1. **PSM-level Conformance** - A Service Provider demonstrating PSM-level Conformance implements one or more of the Systems Modeling API and Services PSMs defined in this specification. For example, a Provider can implement the REST/HTTP PSM, the OSLC PSM, or both. PSM-level conformance of Service Providers ensures interoperability of Service Consumers using the PSM across the different Service Providers. See **Clause 8**.

2. **PIM-level Conformance** - A Service Provider demonstrating PIM-level Conformance implements a PSM that is not defined in this specification but is based on Systems Modeling API and Services PIM defined in this specification. The Service Provider shall define the PSM and the mapping from PIM to PSM with the goal that the new PSM may become part of future versions of this specification. See **Clause 7**.

A Service Provider tool must demonstrate conformance to one or more services, as described below.

1. **ProjectService Conformance** - A Service Provider must implement all the operations in the ProjectService, and demonstrate that the implementation successfully passes all the ProjectService Conformance Test Cases (see **A.1**) and Cross-Cutting Conformance Test Cases (see **A.7**).

2. **ElementNavigationService Conformance** - A Service Provider must implement all the operations in the ElementNavigationService, and demonstrate that the implementation successfully passes all
3. **ProjectDataVersioningService Conformance** - A Service Provider must implement all the operations in the ProjectDataVersioningService, and demonstrate that the implementation successfully passes all the ProjectDataVersioningService Conformance Test Cases (see A.3) and Cross-Cutting Conformance Test Cases (see A.7).
   
   1. **Derived Property Conformance** - A Service Provider conformant to the ProjectDataVersioningService may optionally demonstrate this additional conformance. The Derived Property Conformance is relevant for project commits where Element (KerML) data is created or updated. In order to demonstrate Derived Property Conformance, a service provider must do the following when ProjectDataVersioningService.createCommit operation is invoked.
      
      1. Compute or verify all derived properties for Element data that is created or updated in the commit.
      2. Include the derived properties in the response, i.e. DataVersion.payload should include derived properties for Element data.
   
4. **QueryService Conformance** - A Service Provider must implement all the operations in the Query Service, and demonstrate that the implementation successfully passes all the QueryService Conformance Test Cases (see A.4) and Cross-Cutting Conformance Test Cases (see A.7).

5. **ExternalRelationshipService Conformance** - A Service Provider must implement all the operations in the ExternalRelationshipService, and demonstrate that the implementation successfully passes all the ExternalRelationshipService Test Cases (see A.5) and Cross-Cutting Conformance Test Cases (see A.7).

6. **ProjectUsageService Conformance** - A Service Provider must implement all the operations in the ProjectUsageService, and demonstrate that the implementation successfully passes all the ProjectUsageService Conformance Test Cases (see A.6) and Cross-Cutting Conformance Test Cases (see A.7).
3 Normative References

[GraphQL] GraphQL
https://graphql.org/

[Gremlin] Gremlin Graph Traversal Machine and Language
https://tinkerpop.apache.org/gremlin.html

[IRI] Internationalized Resource Identifiers (IRI)
https://www.w3.org/International/articles/idn-and-iri/

[KerML] Kernel Modeling Language (KerML), Version 1.0
(as submitted with this proposed specification)

[MOFVD] MOF2 Versioning and Development Lifecycle (MOFVD™), Version 2.0
https://www.omg.org/spec/MOFVD/2.0

[OpenAPI] OpenAPI Specification
https://www.openapis.org/

[OSLC] Open Services for Lifecycle Collaboration (OSLC)
http://open-services.net/

[QVT] MOF Query/View/Transformation (QVT™), Version 1.3
https://www.omg.org/spec/QVT/1.3

[SEBoK] Systems Engineering Body of Knowledge (SEBoK)
www.sebokwiki.org

https://www.incose.org/products-and-publications/se-handbook

[SMOF] MOF Support for Semantic Structures (SMOF™), Version 1.0
https://www.omg.org/spec/SMOF/1.0

[SPARQL] SPARQL Query Language for RDF
https://www.w3.org/TR/rdf-sparql-query/

https://www.iso.org/standard/63555.html

[STEP] ISO 10303-233:2012 (STEP)
https://www.iso.org/standard/55257.html

[SysML] OMG Systems Modeling Language (SysML®), Version 2.0
(as submitted with this proposed specification)

[UML] Unified Modeling Language (UML), Version 2.5.1
https://www.omg.org/spec/UML/2.5.1

[UUID] Universally Unique IDentifier (UUID) URN Namespace
4 Terms and Definitions

Various terms and definitions are specified throughout the body of this specification.
5 Symbols

There are no special symbols defined in this specification.
6 Introduction

6.1 API and Services Architecture

The Systems Modeling API and Services includes the following.

(1) **Platform-Independent Model (PIM)** provides a service specification independent of any platform or technology. This specification defines each of the services and their operations with inputs and outputs. The PIM serves as the logical API model.

(2) **Platform-Specific Models (PSMs)** are bindings of the PIM using a particular technology, such as REST/HTTP, SOAP, Java, and .NET. Multiple platform-specific models can exist for a given PIM. Two PSMs are provided in this specification:

- REST/HTTP PSM - a binding of the PIM as a REST/HTTP API using OpenAPI specification.
- OSLC PSM - a binding of the PIM as services based on the OSLC standard.

For each PSM, a mapping is defined. This mapping is used to generate the PSM from the PIM.

Fig. 2 illustrates the PIM, PSMs, Service Providers that implement API PSMs, and Service Consumers that consume the API PSMs from multiple Providers.

Figure 2. API and Services Architecture

Service specifications in the PIM do not prescribe or constrain the architecture of the Service Providers. For example, Service Providers with file-based, 3-tier application-based, or federated microservices-based architectures can all implement one or more PSMs derived from the same service specifications (PIM).

Service Consumers that use a specific PSM should be interoperable across multiple Service Providers that implement that PSM without requiring any modification in the consumer.
Figure 3. Use of PIM and PSMs by Providers and Consumers

Fig. 3 illustrates the role of PIM and PSMs in the context of Systems Modeling API and Services providers and consumers. The Systems Modeling API and Services, version 1.0, includes two PSMs, specifically the REST/HTTP PSM and OSLC 3.0 PSM.

A System Modeling API and Services provider implements either or both the PSMs using its native technology stack, such as databases and web-service frameworks. Service consumers, such as those used for programmatic, graphical, or textual authoring, navigation, and querying data use the PSMs (e.g. REST/HTTP API), agnostic of the native technology stack of the providers.

The choice of REST/HTTP PSM is key. Most modern programming languages provide libraries for consuming REST/HTTP APIs. Enterprise applications, written in any modern programming language, can consume the standard Systems Modeling API and Services, and interoperate with multiple providers.

6.2 Document Conventions

The following stylistic conventions are applied in the presentation of the Platform Independent Model (PIM) of the Systems Modeling API and Services.

Service definitions

1. Names of classes representing services start with an uppercase letter and use the camel case notation, such as ElementNavigationService.
2. Names of operations representing the API calls available for each service start with a lowercase letter and are italicized, such as getElementById.

Input and output data

1. Names of classes representing data that is the input or output of services start with an uppercase letter, such as Project and Data.
2. Names of attributes representing the details of the data that is the input or output of services start with a lowercase letter and are italicized, such as identifier.
The services and operations in the PIM are presented using class diagrams and tables with descriptions of each operation.

The input and output data for services in the PIM are presented using class diagrams followed by detailed descriptions.

### 6.3 Document Organization

The rest of this document is organized into two major clauses.

- Clause 7 - Platform Independent Model (PIM) of the Systems Modeling API and Services
- Clause 8 - Platform Specific Models (PSMs) of the Systems Modeling API and Services
  - 8.1 - REST/HTTP PSM
  - 8.2 - OSLC PSM

These clauses are followed by two annexes.

- Annex A defines the suite of conformance tests that must be used to demonstrate the conformance of Service Providers to this specification - see Clause 2.
- Annex B includes the following.
  - Examples of requests and responses for the REST/HTTP API endpoints, and
  - Cookbook with a collection of recipes, as Jupyter notebooks, demonstrating patterns and examples for using the Systems Modeling API and Services

### 6.4 Acknowledgements

The primary authors of this specification document, the PIM, and the REST/HTTP PSM are:

- Manas Bajaj, Intercax LLC
- Ivan Gomes, Twingineer LLC

The primary authors of the OSLC PSM are:

- David Honey, IBM
- Jad El-Khoury, KTH Royal Institute of Technology
- Jim Amsden, IBM

The specification was formally submitted for standardization by the following organizations:

- 88Solutions Corporation
- Dassault Systèmes
- GISe e.V.
- IBM
- INCOSE
- Intercax LLC
- Lockheed Martin Corporation
- Model Driven Solutions, Inc.
- PTC
- Simula Research Laboratory AS

However, work on the specification was also supported by over 200 people in 80 organizations that participated in the SysML v2 Submission Team (SST). The following individuals had leadership roles in the SST:

- Manas Bajaj, Intercax LLC (API and services development lead)
• Yves Bernard, Airbus (v1 to v2 transformation co-lead)
• Bjorn Cole, Lockheed Martin Corporation (metamodel development co-lead)
• Sanford Friedenthal, SAF Consulting (SST co-lead, requirements V&V lead)
• Charles Galey, Lockheed Martin Corporation (metamodel development co-lead)
• Karen Ryan, Siemens (metamodel development co-lead)
• Ed Seidewitz, Model Driven Solutions (SST co-lead, pilot implementation lead)
• Tim Weilkiens, oose (v1 to v2 transformation co-lead)

The specification was prepared using CATIA No Magic modeling tools and the OpenMBEE system for model publication ([http://www.openmbee.org](http://www.openmbee.org)), with the invaluable support of the following individuals:

• Tyler Anderson, No Magic/Dassault Systèmes
• Christopher Delp, Jet Propulsion Laboratory
• Ivan Gomes, Jet Propulsion Laboratory
• Robert Karban, Jet Propulsion Laboratory
• Christopher Klotz, No Magic/Dassault Systèmes
• John Watson, Lightstreet consulting

The following individuals made significant contributions to the API and Services pilot implementation developed by the SST in conjunction with the development of this specification:

• Manas Bajaj, Intercax LLC
• Ivan Gomes, Twingineer LLC
• Brian Miller, Intercax LLC
7 Platform Independent Model (PIM)

7.1 API Model

7.1.1 Record

Figure 4. Types of Records

**Record** - A Record represents any data that is consumed (input) or produced (output) by the Systems Modeling API and Services. A Record is an abstract concept from which other concrete concepts inherit. A Record has the following attributes:

- *id* is the UUID assigned to the record
- *resourceIdentifier* is an IRI for the record
- *alias* is a collection of other identifiers for this record, especially if the record was created or represented in other software applications and systems
- *humanIdentifier* is a human-friendly unique identifier for this record
- *description* is a statement that provides details about the record.
Figure 5. Project Data Versioning API Model

The class diagram above presents concepts related to ProjectDataVersioningService.

**Data** - Data represents any entity that can be created, updated, deleted, and queried by the Systems Modeling API and Services. In the PIM, Data is represented as an Interface that is realized by the following concepts in the scope of Systems Modeling API and Services.

- Element, root metaclass in the KerML abstract syntax [KerML]
- ExternalData
- ExternalRelationship
- ProjectUsage

Each realization of Data must implement the `getId()` operation that provides a valid UUID.

**DataIdentity** - DataIdentity is a subclass of Record that represents a unique, version-independent representation of Data through its lifecycle. A DataIdentity is associated with 1 or more DataVersion records that represent different versions of the same Data. A DataIdentity record has the following attribute:

- version is the set of DataVersion records representing all versions of the given DataIdentity

**DataVersion** - DataVersion is a subclass of Record that represents Data at a specific version in its lifecycle. A DataVersion record is associated with only one (1) DataIdentity record. DataVersion serves as a wrapper for Data (payload) in the context of a Commit in a Project. Different versions of the same Data, identified by the same UUID values returned by `Data.getId()`, are represented in the following manner:

- One (1) DataIdentity record is created for all versions of the same Data, where DataIdentity.id returns the same UUID value as `Data.getId()`
• A DataVersion record is created for each version of Data, where:
  ◦ DataVersion.payload is set to Data
  ◦ DataVersion.identity is set to the DataIdentity common to all versions of the same Data
  ◦ DataVersion.id is set to a new, randomly generated UUID for the specific DataVersion record

DataVersion record has the following additional attributes:

• commit is the Project commit at which the wrapped Data (payload) was created, modified, or deleted
• project is a derived attribute referencing the owning Project

Project - Project is a subclass of Record that represents a container for other Records and an entry point for version management and data navigation. The Project record has the following attributes:

• name is the name of the Project
• commits is the set of all commits in the Project
• commitReferences is the set of all commit references in the Project
• branches is the set of all the branches in the Project and a subset of commitReferences
• defaultBranch is the default branch in the Project and a subset of branches
• tags is the set of all the tags in the Project and a subset of commitReferences
• queries is the set of Query records owned by the Project. Each Query record represents a saved query for the given Project. See Query for details.

A Project also represents a permission target at which access and authorization controls may be applied to teams associated with a Project.

ProjectUsage - ProjectUsage is a realization of Data that represents the use of a Project in the context of another Project. ProjectUsage has the following attributes:

• usedProjectCommit references the Commit of the Project being used

Commit - Commit is a subclass of Record that represents the changes made to a Project at a specific point in time in its lifecycle, such as the creation, update, or deletion of Data in a Project. A Project has 0 or more Commits. A Commit has the following attributes:

• created is the timestamp at which the Commit was created
• owningProject is the Project that owns the Commit.
• previousCommits is the set of immediately preceding Commits
• change is the set of DataVersion records representing Data that is created, updated, or deleted in the Commit
• versionedData is a derived attribute representing the cumulative set of DataVersion records in a Project at the given Commit

Commits are immutable. For a given Commit record, the value of Commit.change cannot be modified after a Commit has been created. If a modification is required, a new Commit record can be created with a different value of Commit.change.

Commits are not destructible. A Commit record cannot be deleted during normal end-user operation. Commits represent the history and evolution of a Project. Deleting and mutating Commit records must be disabled for normal end-user operations to preserve Project history.

1Note: A Service Provider may provide administrative functions to modify the Commit graph of a Project where commits may need to be deleted for special cases but this is not considered a normal end-user operation. One such special case is when multiple Commits are squashed - a new Commit representing the union of changes in given Commits is created and then the given Commits need to be deleted.
CommitReference - CommitReference is an abstract subclass of Record that references a specific Commit (CommitReference.referencedCommit). Project.commits is the set of all the Commit records for a given Project. Project.commitReferences identifies specific Commit records in a Project that provide the context for navigating the Data in a Project. Two special types of CommitReference are Branch and Tag, as described below.

A CommitReference has the following additional attributes.

- `created` is the timestamp at which the CommitReference was created
- `name` is the name of the CommitReference, e.g. name of the Branch or the Tag
- `owningProject` is the Project that owns the given CommitReference

Branch - Branch is an indirect subclass of Record (via CommitReference) that represents an independent line of development in a Project. A Project can have 1 or more Branches. When a Project is created, a default Branch is also created. The default Branch of a Project can be changed, and a Project can have only 1 default Branch.

A Branch is a type of CommitReference. A Branch is a pointer to a Commit (`Branch.head`). The Commit history of a Project on a given Branch can be computed by recursively navigating `Commit.previousCommits`, starting from the head Commit of the Branch (`Branch.head`). A Branch has the following additional attributes:

- `head` is the Commit to which the Branch is currently pointing. It represents the latest state of the Project on the given Branch.
- `owningProject` is the Project that owns the given Branch

Branches are mutable. Since a Branch is a pointer to a Commit, it can be updated to point to a different Commit. If a new Commit is created on a Project Branch, the value of `Branch.head` refers to that new Commit.

Branches are destructible under normal end-user operation. Branches can be deleted and merged with other branches.

Tag - Tag is an indirect subclass of Record (via Commit Reference) used for annotating specific Commits-of-interest during Project development, such as for representing Project milestones, releases, baselines, or snapshots. A Project can have 0 or more Tags.

A Tag is a type of CommitReference. A Tag is a pointer to a Commit (`Tag.taggedCommit`). A Tag has the following attributes:

- `taggedCommit` is the Commit to which the Tag is pointing. It represents a Commit-of-interest in the owning Project.
- `owningProject` is the Project that owns the given Tag

Tags are immutable. `Tag.taggedCommit` cannot be modified after a Tag record has been created. If `Tag.taggedCommit` needs to be modified to refer to a different Commit record, then the existing Tag can be deleted and a new Tag can be created with the same name and description.

Tags are destructible under normal end-user operation.

The table below summarizes the mutability and destruction semantics of Commit, Branch, and Tag for normal end-user operations.

<table>
<thead>
<tr>
<th>Type of Record</th>
<th>Mutable</th>
<th>Destructible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Branch</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Systems Modeling API and Services v1.0 Beta 1
7.1.3 ExternalData and ExternalRelationship

The class diagram above presents concepts related to ExternalRelationship Service.

**ExternalRelationship** - ExternalRelationship is a realization of Data, and represents the relationship between a KerML Element [KerML] in a provider tool or repository to ExternalData in another tool or repository. The ExternalData may be a KerML Element or a non-KerML Element. A hyperlink between a KerML Element to a web resource is the most primitive example of an ExternalRelationship. An ExternalRelationship has the following attributes:

- **specification** is the formal representation of the semantics of the ExternalRelationship. The specification can be a collection of mathematical expressions. For example, an ExternalRelationship can be defined to map the attributes of a KerML Element to the attributes of an ExternalData. In this case, the specification would contain mathematical expressions, such as equations, representing the mapping. This is an optional attribute.
- **language** is the name of the expression language used for the specification. This is an optional attribute.

**ExternalData** - ExternalData is a realization of Data, and represents a resource external to a given tool or repository. ExternalData is defined only for the purpose of defining an ExternalRelationship. An ExternalData has the following additional attributes:

- **resourceIdentifier** is the IRI of the resource represented by the ExternalData.
7.1.4 Query

Figure 7. Query API Model

The class diagram above presents concepts related to the Query service.

Query - Query is a subclass of Record that represents a precise and language-independent request for information retrieval using the Systems Modeling API and Services. Query can be mapped to commonly used query languages, such as SQL, Gremlin, GraphQL, and SPARQL.

A Query record has the following attributes:

- *name* is the name of the Query
• **select** is a list of properties of Data (or its realizations) that will be included for each Data object in the query response. If no properties are specified, then all the properties will be included for each Data object in the query response.

• **scope** is a list of Data objects that define the scope context for query execution. The default scope of a Query is the owning Project.

• **where** is a Constraint that represents the conditions that Data objects in the query response must satisfy

• **orderBy** is a list of properties of Data (or its realizations) that are used for sorting the Data objects in the query response. The order of properties in the list governs the sorting order.

**Constraint** - Constraint is an abstract concept that represents conditions that must be satisfied by Data objects in the query response.

**PrimitiveConstraint** is a concrete subtype of Constraint that represents simple conditions that can be modeled using the *property-operator-value* tuple, e.g. `mass <= 4 kg`, or *type instanceOf Generalization*. A PrimitiveConstraint has the following attributes:

- **property** is a property of Data (or its realizations) that is being constrained
- **operator** is of type Enumeration whose literals are mathematical operators, as shown in the figure above
- **value** is a list of primitive objects, such as String, Boolean, Integer, Double, and UUID
- **inverse** is of type Boolean. If true, a logical NOT operator is applied to the PrimitiveConstraint.

**CompositeConstraint** is a concrete subtype of Constraint that represents complex conditions composed of two or more Constraints using logical AND or OR operator. CompositeConstraint has the following attributes:

- **constraint** is the set of Constraints being composed
- **operator** is the logical operator for composing the Constraints

### 7.2 API Services

#### 7.2.1 ProjectService

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProjects()</td>
<td>Get all projects.</td>
</tr>
<tr>
<td>getProjectById( projectId : UUID )</td>
<td>Get project with the given id (projectId).</td>
</tr>
<tr>
<td>createProject( name : String, description : String [0..1] )</td>
<td>Create a new project with the given name and description (optional).</td>
</tr>
<tr>
<td>updateProject( projectId : UUID, name : String [0..1], description : String [0..1], defaultBranch : Branch [0..1] )</td>
<td>Update the project with the given id (projectId).</td>
</tr>
<tr>
<td>deleteProject( projectId : UUID )</td>
<td>Delete the project with the given id (projectId).</td>
</tr>
</tbody>
</table>

**Figure 8. ProjectService Operations**

**Table 1. Operations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateProject</td>
<td>Update the project with the given id (projectId).</td>
</tr>
<tr>
<td>getProjectById</td>
<td>Get project with the given id (projectId).</td>
</tr>
<tr>
<td>createProject</td>
<td>Create a new project with the given name and description (optional).</td>
</tr>
<tr>
<td>deleteProject</td>
<td>Delete the project with the given id (projectId).</td>
</tr>
<tr>
<td>getProjects</td>
<td>Get all projects.</td>
</tr>
</tbody>
</table>
### 7.2.2 ElementNavigationService

<table>
<thead>
<tr>
<th>operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getElements( project : Project, commit : Commit ) : Element [0..*]</code></td>
</tr>
<tr>
<td><code>getElementById( project : Project, commit : Commit, elementId : UUID ) : Element [0..1]</code></td>
</tr>
<tr>
<td><code>getRelationshipsByRelatedElement( project : Project, commit : Commit, elementId : UUID, direction : Direction ) : Relationship [0..*]</code></td>
</tr>
<tr>
<td><code>getRootElements( project : Project, commit : Commit ) : Element [0..*]</code></td>
</tr>
</tbody>
</table>

#### Figure 9. ElementNavigationService Operations

Element is the root metaclass in the KerML abstract syntax [KerML]. Relationship is a subtype of Element. Both Element and Relationship realize the Data interface defined in the API Model (refer to 7.1.2 - Project Data Versioning).

#### Table 2. Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getElements</code></td>
<td>Get all the elements in a given project at the given commit.</td>
</tr>
<tr>
<td><code>getElementById</code></td>
<td>Get element with the given id (elementId) in the given project at the given commit.</td>
</tr>
<tr>
<td><code>getRelationshipsByRelatedElement</code></td>
<td>Get relationships that are incoming, outgoing, or both relative to the given related element.</td>
</tr>
<tr>
<td><code>getRootElements</code></td>
<td>Get all the root elements in the given project at the given commit.</td>
</tr>
</tbody>
</table>
7.2.3 ProjectDataVersioningService

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCommitChangeById</td>
<td>Get the change with the given id (changeId) in the given commit of the given project. The changeId is the id of the DataVersion that changed in the commit.</td>
</tr>
<tr>
<td>getBranchById</td>
<td>Get the branch with the given id (branchId) in the given project.</td>
</tr>
<tr>
<td>getTagById</td>
<td>Get the tag with the given id (tagId) in the given project.</td>
</tr>
<tr>
<td>getCommits</td>
<td>Get all the commits in the given project.</td>
</tr>
<tr>
<td>getBranches</td>
<td>Get all the branches in the given project.</td>
</tr>
</tbody>
</table>

Figure 10. ProjectDataVersioningService Operations

Table 5. Operations
<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCommit</td>
<td>Create a new commit with the given change (collection of DataVersion records) in the given branch of the project. If the branch is not specified, the default branch of the project is used. Commit.change should include the following for each Data object that needs to be created, updated, or deleted in the new commit. (1) Creating Data - Commit.change should include a DataVersion record with DataVersion.payload populated with the Data being created. DataVersion.identity is not provided, thereby indicating that a new DataIdentity needs to be created in the new commit. (2) Updating Data - Commit.change should include a DataVersion record with DataVersion.payload populated with the updated Data. DataVersion.identity should be populated with the DataIdentity for which a new DataVersion record will be created in the new commit. (3) Deleting Data - Commit.change should include a DataVersion record with DataVersion.payload not provided, thereby indicating deletion of DataIdentity in the new commit. DataVersion.identity should be populated with the DataIdentity that will be deleted in the new commit. When a DataIdentity is deleted in a commit, all its versions (DataVersion) are also deleted, and any references from other DataIdentity are also removed to maintain data integrity. In addition, for Element Data (KerML), deletion of an Element must also result in deletion of incoming Relationships. When Element Data (KerML) is created or updated, derived properties must be computed or verified if the API provider claims Derived Property Conformance.</td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>Documentation</strong></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td><strong>diffCommits</strong></td>
<td>Get the difference between two commits - compareCommit and baseCommit. The set of all DataVersion records in a project at a given commit is accessible as Commit.versionedData. From a set theoretic perspective, this operation gets compareCommit.versionedData - baseCommit.versionedData and returns a DataDifference object with baseData and compareData for each difference. If any data is present in the compareCommit but absent in the baseCommit, DataDifference.compareData will include the corresponding DataVersion and DataDifference.baseData will be empty. If any data is absent in the compareCommit but present in the baseCommit, DataDifference.compareData will be empty and DataDifference.baseData will include the corresponding DataVersion. If any data is present in both but different in the compareCommit and baseCommit, DataDifference.compareData and DataDifference.baseData will include the corresponding DataVersion records.</td>
</tr>
<tr>
<td><strong>mergeIntoBranch</strong></td>
<td>Merge the given commits (commitsToMerge) in the given branch (baseBranch). The commits included in commitsToMerge may be commits referenced by a CommitReference, such as Branch.head or Tag.taggedCommit, or any other commit in the owning project (Project.commits). This operation returns a MergeResult which will include either of the following: (1) commit after the merge operation if successful, or (2) a set of DataIdentity records representing the merge conflicts if the merge operation is unsuccessful. Two optional inputs may be provided: (1) resolution as a set of Data that will resolve the merge conflicts, and (2) description of the merged commit if this operation is successful.</td>
</tr>
<tr>
<td><strong>getCommitById</strong></td>
<td>Get the commit with the given id (commitId) in the given project.</td>
</tr>
<tr>
<td><strong>deleteBranch</strong></td>
<td>Delete the branch with the given id (branchId) in the given project.</td>
</tr>
<tr>
<td><strong>getDefaultBranch</strong></td>
<td>Get the default branch of the given project.</td>
</tr>
<tr>
<td><strong>deleteTag</strong></td>
<td>Delete the tag with the given id (tagId) in the given project.</td>
</tr>
<tr>
<td><strong>setDefaultBranch</strong></td>
<td>Set the branch with the given branchId as the default branch of the given project.</td>
</tr>
<tr>
<td><strong>getHeadCommit</strong></td>
<td>Get the head commit of the given branch in the given project. If the branch is not specified, the default branch of the project is used.</td>
</tr>
</tbody>
</table>
### 7.2.4 QueryService

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTaggedCommit</td>
<td>Get the tagged commit of the given tag in the given project.</td>
</tr>
<tr>
<td>getCommitChange</td>
<td>Get the change in the given commit of the given project.</td>
</tr>
<tr>
<td>getTags</td>
<td>Get all the tags in the given project.</td>
</tr>
<tr>
<td>createTag</td>
<td>Create a new tag with the given name (tagName) in the given project, and set the taggedCommit of the new tag as the given commit (taggedCommit).</td>
</tr>
<tr>
<td>createBranch</td>
<td>Create a new branch with the given name (branchName) in the given project, and set the head of the new branch as the given commit (head).</td>
</tr>
</tbody>
</table>

**Figure 11. QueryService Operations**

**Table 6. Operations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getQueryById</td>
<td>Get the query with the given id (queryId) in the given project.</td>
</tr>
<tr>
<td>getQueries</td>
<td>Get all the queries in the given project.</td>
</tr>
<tr>
<td>executeQueryById</td>
<td>Execute the query with the given id in the owning project (Query.project) at the given commit. If the commit is not specified, then the head commit of the default branch of the project will be used.</td>
</tr>
<tr>
<td>createQuery</td>
<td>Create a query in the given project with the given inputs.</td>
</tr>
<tr>
<td>deleteQuery</td>
<td>Delete the query with the given id (queryId) in the given project.</td>
</tr>
<tr>
<td>updateQuery</td>
<td>Update the given query (updateQuery) in the given project.</td>
</tr>
<tr>
<td>executeQuery</td>
<td>Execute the given query in the owning project (Query.project) at the given commit. If the commit is not specified, then the head commit of the default branch of the project will be used.</td>
</tr>
</tbody>
</table>
7.2.5 ExternalRelationshipService

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getExternalRelationshipById</td>
<td>Get the external relationship with the given id (externalRelationshipId).</td>
</tr>
<tr>
<td>getExternalRelationshipsByElement</td>
<td>Get all the external relationships in the given project at the given commit, where the id of elementEnd of the external relationship is the given elementId.</td>
</tr>
<tr>
<td>getExternalRelationships</td>
<td>Get all the external relationships in a given project at a given commit.</td>
</tr>
</tbody>
</table>

Figure 12. ExternalRelationshipService Operations

7.2.6 ProjectUsageService

<table>
<thead>
<tr>
<th>Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>createProjectUsage</td>
<td>Create a new project usage in the given project at the head commit of the given branch. This operation returns a new commit that includes the new project usage, and sets the head of the given branch to the new commit. If a project branch is not given, then the default branch of the project will be used.</td>
</tr>
<tr>
<td>deleteProjectUsage</td>
<td>Deletes the project usage with the given id (projectUsageId) from the given project at the head commit of the given branch. This operation returns a new commit where the given project usage does not exist, and sets the head of the given branch to the new commit. If a project branch is not given, then the default branch of the project will be used.</td>
</tr>
<tr>
<td>getProjectUsages</td>
<td>Get all the project usages in the given project at the given commit.</td>
</tr>
</tbody>
</table>

Figure 13. ProjectUsageService Operations
8 Platform Specific Models (PSMs)

8.1 REST/HTTP PSM

8.1.1 Overview

The REST/HTTP Platform-Specific Model (PSM) for the Systems Modeling API and Services is described using OpenAPI Specification (OAS) 3.1 and is included with this specification. The REST/HTTP PSM is described in the following sections.

- **PIM API Model - REST/HTTP PSM Model Mapping:** This section presents the mapping from the PIM API Model concepts to the JSON Models in the REST/HTTP PSM (OpenAPI specification).
- **PIM API Services - REST/HTTP PSM Endpoints Mapping:** This section presents the mapping from the PIM API Service definitions and operations to the API endpoints in the REST/HTTP PSM (OpenAPI specification).

8.1.2 PIM API Model - REST/HTTP PSM Model Mapping

The table below presents the mapping from the PIM API Model concepts to the JSON Models in the REST/HTTP PSM (OpenAPI specification).

<table>
<thead>
<tr>
<th>PIM Concept</th>
<th>REST/HTTP PSM Model (JSON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Project</td>
</tr>
<tr>
<td>Commit</td>
<td>Commit</td>
</tr>
<tr>
<td>Tag</td>
<td>Tag</td>
</tr>
<tr>
<td>Branch</td>
<td>Branch</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>DataIdentity</td>
<td>DataIdentity</td>
</tr>
<tr>
<td>DataVersion</td>
<td>DataVersion</td>
</tr>
<tr>
<td>Element</td>
<td>Element</td>
</tr>
<tr>
<td>Relationship</td>
<td>Relationship</td>
</tr>
<tr>
<td>ExternalData</td>
<td>ExternalData</td>
</tr>
<tr>
<td>ExternalRelationship</td>
<td>ExternalRelationship</td>
</tr>
<tr>
<td>ProjectUsage</td>
<td>ProjectUsage</td>
</tr>
<tr>
<td>Query</td>
<td>Query</td>
</tr>
<tr>
<td>PrimitiveConstraint</td>
<td>PrimitiveConstraint</td>
</tr>
<tr>
<td>CompositeConstraint</td>
<td>CompositeConstraint</td>
</tr>
<tr>
<td>DataDifference</td>
<td>DataDifference</td>
</tr>
<tr>
<td>MergeResult.mergeCommit</td>
<td>Commit</td>
</tr>
<tr>
<td>MergeResult.conflict</td>
<td>DataIdentity [0..*]</td>
</tr>
</tbody>
</table>
# 8.1.3 PIM API Services - REST/HTTP PSM Endpoints Mapping

The table below presents the mapping between the PIM Services to the REST/HTTP PSM Endpoints. This is followed by a detailed description of the pagination strategy used by the REST/HTTP PSM.

## Table 10. PIM to REST / HTTP PSM Mapping

<table>
<thead>
<tr>
<th>PIM Service</th>
<th>REST / HTTP PSM Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProjectService</strong></td>
<td></td>
</tr>
<tr>
<td>createProject</td>
<td>POST /projects</td>
</tr>
<tr>
<td>getProjects</td>
<td>GET /projects</td>
</tr>
<tr>
<td>getProjectById</td>
<td>GET /projects/{projectId}</td>
</tr>
<tr>
<td>updateProject</td>
<td>PUT /projects/{projectId}</td>
</tr>
<tr>
<td>deleteProject</td>
<td>DELETE /projects/{projectId}</td>
</tr>
<tr>
<td><strong>ElementNavigationService</strong></td>
<td></td>
</tr>
<tr>
<td>getElements</td>
<td>GET /projects/{projectId}/commits/{commitId}/elements</td>
</tr>
<tr>
<td>getElementById</td>
<td>GET /projects/{projectId}/commits/{commitId}/elements/{elementId}</td>
</tr>
</tbody>
</table>
| getRelationshipsByRelatedElement| GET /projects/{projectId}/commits/{commitId}/elements/{relatedElementId}/relationships  
  - *direction* query parameter with allowable values \{in, out, both\} |
| getRootElements                 | GET /projects/{projectId}/commits/{commitId}/roots |
| **ProjectDataVersioningService**|                          |
| getCommits                      | GET /projects/{projectId}/commits |
| getHeadCommit                   | GET /projects/{projectId}/branches/{branchId} returns the branch with the given branch ID. If the branch ID is not provided, then the ID of the default branch of the project is used. Use the following steps to get the ID of the default branch.  
  - GET /projects/{projectId} return the project with the given project ID  
  - Project.defaultBranch provides the ID of the default branch of the project  
  - Branch.head provides the ID of the head commit of the branch  
  - GET /projects/{projectId}/commits/{commitId} returns the head commit with the the given commit ID |
<p>| getCommitById                   | GET /projects/{projectId}/commits/{commitId} |</p>
<table>
<thead>
<tr>
<th>PIM Service</th>
<th>REST / HTTP PSM Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCommit</td>
<td>POST /projects/{projectId}/commit</td>
</tr>
<tr>
<td></td>
<td>• The body of the POST request is a CommitRequest.</td>
</tr>
<tr>
<td></td>
<td>• The content of CommitRequest.change for creating, updating, and deleting Data maps directly to the corresponding PIM operation (createCommit), and is described below.</td>
</tr>
<tr>
<td></td>
<td>• For creating new Data, CommitRequest.change should include a DataVersion where DataVersion.payload includes the Data being created and DataVersion.identity is not specified.</td>
</tr>
<tr>
<td></td>
<td>• For updating existing Data, CommitRequest.change should include a DataVersion where DataVersion.payload includes the updated Data, and DataVersion.identity is the DataIdentity for which a new DataVersion will be created in the commit.</td>
</tr>
<tr>
<td></td>
<td>• For deleting existing Data, CommitRequest.change should include a DataVersion where DataVersion.payload is not specified, and DataVersion.identity is the DataIdentity that will be deleted in the commit.</td>
</tr>
<tr>
<td>getCommitChange</td>
<td>GET /projects/{projectId}/commits/{commitId}/changes</td>
</tr>
<tr>
<td>getCommitChangeById</td>
<td>GET /projects/{projectId}/commits/{commitId}/changes/{changeId}</td>
</tr>
<tr>
<td>getBranches</td>
<td>GET /projects/{projectId}/branches</td>
</tr>
<tr>
<td>getBranchById</td>
<td>GET /projects/{projectId}/branches/{branchId}</td>
</tr>
<tr>
<td>getDefaultBranch</td>
<td>• GET /projects/{projectId} returns a Project with the given ID (projectId)</td>
</tr>
<tr>
<td></td>
<td>• Project.defaultBranch provides the ID of the default branch of the project</td>
</tr>
<tr>
<td>setDefaultBranch</td>
<td>• PUT /projects/{projectId}</td>
</tr>
<tr>
<td></td>
<td>• The body of the PUT request is a ProjectRequest. Set the ID of the new default branch as ProjectRequest.defaultBranch in the body.</td>
</tr>
<tr>
<td>createBranch</td>
<td>POST /projects/{projectId}/branches</td>
</tr>
<tr>
<td>deleteBranch</td>
<td>DELETE /projects/{projectId}/branches/{branchId}</td>
</tr>
<tr>
<td>getTags</td>
<td>GET /projects/{projectId}/tags</td>
</tr>
<tr>
<td>getTagById</td>
<td>GET /projects/{projectId}/tags/{tagId}</td>
</tr>
<tr>
<td>PIM Service</td>
<td>REST / HTTP PSM Endpoint</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getTaggedCommit</td>
<td>• GET /projects/{projectId}/tags/{tagId} returns the tag with the given ID (tagId)</td>
</tr>
<tr>
<td></td>
<td>• Tag.taggedCommit provides the ID of the tagged commit</td>
</tr>
<tr>
<td></td>
<td>• GET /projects/{projectId}/commits/{commitId} returns the tagged commit given its ID (see the previous step)</td>
</tr>
<tr>
<td>createTag</td>
<td>POST /projects/{projectId}/tags</td>
</tr>
<tr>
<td>deleteTag</td>
<td>DELETE /projects/{projectId}/tags/{tagId}</td>
</tr>
<tr>
<td>mergeIntoBranch</td>
<td>POST /projects/{projectId}/branches/{targetBranchId}/merge</td>
</tr>
<tr>
<td>diffCommits</td>
<td>GET /projects/{projectId}/commits/{compareCommitId}/diff</td>
</tr>
</tbody>
</table>

**QueryService**

| getQueries          | GET /projects/{projectId}/queries                                                                              |
| getQueryById        | GET /projects/{projectId}/queries/{queryId}                                                                  |
| createQuery         | POST /projects/{projectId}/queries                                                                           |
| updateQuery         | PUT /projects/{projectId}/queries/{queryId}                                                                  |
| deleteQuery         | DELETE /projects/{projectId}/queries/{queryId}                                                                |
| executeQueryById    | GET /projects/{projectId}/queries/{queryId}/results                                                          |
| executeQuery        | GET /projects/{projectId}/query-results                                                                       |
|                     | POST /projects/{projectId}/query-results                                                                      |
|                     | Either the GET or the POST endpoint may be used. The POST endpoint is provided for compatibility with clients that don't support GET requests with a body |

**ExternalRelationshipService**

<p>| getExternalRelationships | POST /projects/{projectId}/queries with QueryRequest JSON model                                              |
|                         | • Query.where is set to a PrimitiveConstraint                                                              |
|                         | • PrimitiveConstraint.property = @type                                                                    |
|                         | • PrimitiveConstraint.value = 'ExternalRelationship'                                                       |
|                         | • PrimitiveConstraint.operator = '='                                                                      |
|                         | • Execute the query with the following request                                                            |
|                         | • GET /projects/{projectId}/queries/{queryId}/results?commitId={commitId}, where {projectId} and {commitId} are the ids of the given Project and Commit |</p>
<table>
<thead>
<tr>
<th>PIM Service</th>
<th>REST / HTTP PSM Endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• POST /projects/{projectId}/queries with QueryRequest JSON model</td>
</tr>
<tr>
<td></td>
<td>◦ Query.where is set to a CompositeConstraint</td>
</tr>
<tr>
<td></td>
<td>◦ CompositeConstraint.constraints includes the following 2 instances of PrimitiveConstraint with the and operator</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint 1</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.property = @type</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.value = 'ExternalRelationship'</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.operator = '='</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint 2</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.property = elementEnd</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.value = {elementId}</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.operator = '='</td>
</tr>
<tr>
<td></td>
<td>• Execute the query with the following request</td>
</tr>
<tr>
<td></td>
<td>◦ GET /projects/{projectId}/queries/{queryId}/results?commitId={commitId}, where {projectId} and {commitId} are the ids of the given Project and Commit</td>
</tr>
<tr>
<td>getExternalRelationshipsByElement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• POST /projects/{projectId}/queries with Query JSON model</td>
</tr>
<tr>
<td></td>
<td>◦ Query.where is set to a CompositeConstraint</td>
</tr>
<tr>
<td></td>
<td>◦ CompositeConstraint.constraints includes the following 2 instances of PrimitiveConstraint with the and operator</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint 1</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.property = @type</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.value = 'ExternalRelationship'</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.operator = '='</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint 2</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.property = @id</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.value = {externalRelationshipId}</td>
</tr>
<tr>
<td></td>
<td>▪ PrimitiveConstraint.operator = '='</td>
</tr>
<tr>
<td></td>
<td>• Execute the query with the following request</td>
</tr>
<tr>
<td></td>
<td>◦ GET /projects/{projectId}/queries/{queryId}/results?commitId={commitId}, where {projectId} and {commitId} are the ids of the given Project and Commit</td>
</tr>
<tr>
<td>getExternalRelationshipsById</td>
<td></td>
</tr>
</tbody>
</table>

**Project Usage Service**
<table>
<thead>
<tr>
<th>PIM Service</th>
<th>REST / HTTP PSM Endpoint</th>
</tr>
</thead>
</table>
| getProjectUsages     | • POST /projects/{projectId}/queries with QueryRequest JSON model
  ◦ Query.where is set to a PrimitiveConstraint
  ◦ PrimitiveConstraint.property = @type
  ◦ PrimitiveConstraint.value = 'ProjectUsage'
  ◦ PrimitiveConstraint.operator = '='
  • Execute the query with the following request
  ◦ GET /projects/{projectId}/queries/{queryId}/results?
    commitId={commitId}, where {projectId} and
    {commitId} are the ids of the given Project and
    Commit |
| createProjectUsage   | • POST /projects/{projectId}/commits?branchId={branchId} with
  CommitRequest JSON model, such that:
  ◦ CommitRequest.change = DataVersion with the
    following inputs
    • DataVersion.payload = ProjectUsage with the
      following inputs
    ◦ ProjectUsage.usedProjectCommit = {commitId} of the Project and
      Commit being used. |
| deleteProjectUsage   | • POST /projects/{projectId}/commits?branchId={branchId} with
  CommitRequest JSON model, such that:
  ◦ CommitRequest.change = DataVersion with the
    following inputs
    • DataVersion.identity = DataIdentity with the
      following inputs
    ◦ DataIdentity.id = {projectUsageId}
    • DataVersion.payload = null |

**Pagination**

The REST/HTTP PSM uses a Cursor-based pagination strategy for the responses received from the GET requests. The following 3 query parameters can be specified in any GET request that returns a collection of records.

1. `page[size]` specifies the maximum number of records that will be returned per page in the response
2. `page[before]` specifies the URL of the page succeeding the page being requested
3. `page[after]` specifies the URL of a page preceding the page being requested

If neither `page[before]` nor `page[after]` is specified, the first page is returned with the same number of records as specified in the `page[size]` query parameter. If the `page[size]` parameter is not specified, then a default page size is used, which can be set by the API provider.

The `Link` header in the response includes links (URLs) to the previous page and the next page, if any, for the given page in the response. The specification of these links is conformant to the IETF Web Linking standard. As an example, the value of the `Link` response header is shown below. The `rel` value associated with each page link
specifies the type of relationship the linked page has with the page returned in the response. Page link specified with rel value as next is the link for the next (or succeeding) page to the page returned in the response, and the page link specified with rel value as prev is the link for the previous (or preceding) page to the page returned in the response.

An example demonstrating the Cursor-based paginated responses received from GET requests to the /projects endpoint is presented here. The term "User" in the example scenario presented below refers to an API user that could be a human user or a software program.

Step 1 - User makes a GET request to the /projects endpoint with page[size] query parameter set to 3. If successful, this request will return the first page with a maximum of 3 project records. The URL for this GET request is shown below.


Step 2 - If there are more than 3 projects in the provider repository, the Link header in the response will provide the URL for the next page with rel value equal to next. The User gathers the link to the next page.

Step 3 - User makes a GET request to the URL for the next page gathered from Step 2. The Link header in the response will provide the URL for the next page with rel value equal to next. Additionally, the Link header will include the URL for the previous page with rel value equal to prev.

Step 4 - User continues Step 3 until the Link header in the response does not include the URL for the next page (rel value as next).

8.2 OSLC 3.0 PSM

8.2.1 Overview

The OSLC Platform-Specific Model (PSM) for the Systems Modeling API and Services is described using OpenAPI Specification (OAS) 2.0 and is included with this specification.

Note that the URLs listed in the OpenAPI Specification are provided as examples only. With OSLC, all URLs are implementation-specific. A OSLC client typically relies on the OSLC discovery mechanism (https://docs.oasis-open-projects.org/oslc-op/core/v3.0/ps01/discovery.html), to determine what services are provided by an OSLC server, as well as the necessary information (such as a service URL) to be able to consume any such service. At the
least, an OSLC client requires a discovery URL to bootstrap this discovery for any particular server. The various approaches for bootstrapping and discovery are further detailed in the OSLC standard.

- 8.2.2 presents a brief introduction to OSLC and its nomenclature.
- 8.2.3 presents the mapping of PIM concepts to OSLC resource types
- 8.2.4 presents the mapping of PIM services and operations to OSLC services

An OSLC implementation may typically need to realize a broader set of services than those defined by the PIM for full integration with other OSLC-compliant systems. Services such as Delegated UI for Selection and Creation, resource UI Preview, authentication, and support for arbitrary queries (beyond those defined in the PIM).

### 8.2.2 OSLC Nomenclature

#### What is OSLC?

Open Services for Lifecycle Collaboration (OSLC) is an open community creating specifications for integrating tools. OSLC specifications allow conforming independent software and product lifecycle tools to integrate their data and workflows in support of end-to-end lifecycle processes. OSLC is based on the W3C Linked Data and the use of RDF to represent artifacts using common vocabularies, and HTTP to discover, create, read, update, and delete such artifacts. For a more comprehensive introduction, see the OSLC Primer and the OSLC specifications at [https://open-services.net/specifications/](https://open-services.net/specifications/).

OSLC servers may support any or all of the following:

1. **Creation factories** for creating a resource of some RDF type associated with the factory. For example, a client might create a new change request by an HTTP POST including the RDF representation of the change request to be created to the URI of a change request creation factory.
2. REST services to read, update, and/or delete resources at the resource's URI.
3. **Query capabilities** that allow OSLC clients to query for resources of an RDF type associated with the query capability. For example, a client might query for change requests by an HTTP GET or POST to the query base URI of a query capability for change requests. See OSLC Query Version 3.0 for further details.
4. **Creation dialog** that allows some other application to embed it in an iFrame of an application dialog that allows a user to fill in information and create a resource of some type associated with the creation dialog.
5. **Selection dialog** that allows an application to display it to select a resource of some type associated with the dialog in order to create and persist a link to that resource in the application.
6. **Resource preview**, such as a pop-up display, that is shown as a rich hover when a user hovers over a link to a resource managed by the OSLC server.

#### OSLC Discovery

OSLC clients and other servers discover the OSLC capabilities offered by a server through OSLC discovery. This allows clients to discover the URIs of creation factories, query capabilities, creation dialogs, and selection dialogs without the need for hard coding or constructing URIs for them. Discovery starts with a known URI for an OSLC Service Provider Catalog. That service provider catalog may reference zero, one, or many service providers. Servers that support the concept of project as a container of resources, often for access control, often define a service provider for each such container. A service provider may declare one or many services, each of which may define the creation factories, query capabilities, creation dialogs, and selection dialogs supported by that service. For more details, see OSLC Core Version 3.0, Part 2: Discovery.

A creation factory for a specific RDF type is discovered by:

1. Start with a known OSLC service provider catalog URI, perform HTTP GET.
2. Get the URIs of service providers from the response.
3. For each service provider, perform HTTP GET.
4. From the response, look for services described in the response data, that declare a creation factory for the RDF type.
5. Get the URI of the creation factory.

A query capability for a specific RDF type is discovered by:

1. Start with a known OSLC service provider catalog URI, perform HTTP GET.
2. Get the URIs of service providers from the response.
3. For each service provider, perform HTTP GET.
4. From the response, look for services described in the response data, that declare a query capability for the RDF type.
5. Get the query base URI of the query capability.

**OSLC Resource Shapes**

Resource shapes specify a standard way (using RDF) of describing resources of specific RDF types and their properties. For more details, see [OSLC Core Version 3.0, Part 6: Resource Shape](#). Resource shapes may be discovered in a number of ways:

1. The definition of a creation factory may reference a resource shape that describes the properties that might be included in the RDF content POSTed to that creation factory.
2. The definition of a query capability may reference a resource shape that describes the properties of the query results and references a shape that describes the properties that might be queried as a condition, or selected to be included in the results.
3. A resource may reference an instance shape that describes the properties of that resource.

**Linked Data Platform Containers**

Linked Data Platform Containers (LDPC) are a way of representing containers as an RDF resource. OSLC specifications specify LPDCs as a container representation. See [W3C Linked Data Platform 1.0](#) for further information.

**OSLC Service Providers**

A "global" service provider will contain one or more services that cross all systems modeling projects. A service provider will be declared for each systems modeling project that provides capabilities specific to that project.

**RDF Media Types**

OSLC recommends that servers support RDF/XML (application/rdf+xml), Turtle (text/turtle, application/x-turtle), and JSON-LD (application/ld+json).

**8.2.3 PIM API Model – OSLC PSM Resource Mapping**

The mapping from the PIM API Model to the OSLC PSM resource types includes the following.

1. Mapping from the KerML and SysML abstract syntax (Element and subtypes) to OSLC resource shapes and vocabulary. The package containing the resulting OSLC resource shapes and vocabulary is included with this specification - see [OSLC_Systems_Modeling_Resource_Shapes_and_Vocabulary.zip](#).
2. Mapping from the API Model concepts to resource types in other OSLC specifications, such as OSLC Configuration Management specification and OSLC Query specification. This mapping is shown in the table below. References to the OSLC specifications mentioned in the table below are as follows.
2. OSLC Query refers to OSLC Query Version 3.0 available at [https://docs.oasis-open-projects.org/oslc-op/query/v3.0/os/oslc-query.html](https://docs.oasis-open-projects.org/oslc-op/query/v3.0/os/oslc-query.html)

<table>
<thead>
<tr>
<th>PIM Concept</th>
<th>OSLC Resource Type (OSLC Specification)</th>
<th>dc:terms:identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
<td>Component (oslc_config:Component) (OSLC Configuration Management)</td>
<td>PIM project id</td>
</tr>
<tr>
<td>Branch</td>
<td>Stream (oslc_config:Stream) (OSLC Configuration Management)</td>
<td>PIM branch id</td>
</tr>
<tr>
<td>Tag</td>
<td>Baseline (oslc_config:Baseline) (OSLC Configuration Management)</td>
<td>PIM tag id</td>
</tr>
<tr>
<td>Commit</td>
<td>Not supported by OSLC Configuration Management.</td>
<td>PIM commit id</td>
</tr>
<tr>
<td>DataIdentity</td>
<td>Concept resource (OSLC Configuration Management)</td>
<td>PIM Data Identity id</td>
</tr>
<tr>
<td>DataVersion</td>
<td>Version resource (oslc_config:VersionResource) (OSLC Configuration Management)</td>
<td>PIM Data Version id</td>
</tr>
<tr>
<td>Query</td>
<td>OSLC Query (OSLC Query)</td>
<td></td>
</tr>
<tr>
<td>PrimitiveConstraint</td>
<td>oslc.where in OSLC Query (OSLC Query)</td>
<td></td>
</tr>
<tr>
<td>CompositeConstraint</td>
<td>oslc.where in OSLC Query (OSLC Query)</td>
<td></td>
</tr>
<tr>
<td>ProjectUsage</td>
<td>Not Available.</td>
<td></td>
</tr>
<tr>
<td>DataDifference</td>
<td>Not Available</td>
<td></td>
</tr>
<tr>
<td>MergeResult</td>
<td>Not Available</td>
<td></td>
</tr>
</tbody>
</table>
### 8.2.4 PIM API Services – OSLC PSM Service Mapping

The table below presents the mapping between the PIM Services to the OSLC 3.0 PSM.

**Table 12. PIM API Services - OSLC Services Mapping**

<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProjectService</strong></td>
<td>In OSLC, a Project is identified by its URL. So simply perform a GET on the Project URL. To search for a project with a given dcterms:identifier (or any other property that is deemed to return a single query result):</td>
</tr>
<tr>
<td></td>
<td>1. Discover a query capability for components (oslc_config:Component). See <a href="#">OSLC Discovery</a> section.</td>
</tr>
<tr>
<td></td>
<td>2. Perform a GET on the query base, specifying a query for the dcterms:identifier in a oslc.where query parameter, and specifying which properties, if any, of the component should be returned in the query result using an oslc.select query parameter.</td>
</tr>
<tr>
<td></td>
<td>Example: Query for components with identifier 123, returning all properties.</td>
</tr>
<tr>
<td>getProjectById</td>
<td>GET queryBaseUri? oslc.where=dcterms%3Aidentifier%3D%22123%22&amp; oslc.select=*</td>
</tr>
<tr>
<td><strong>createProject</strong></td>
<td>1. Discover a creation factory for components (oslc_config:Component). See <a href="#">OSLC Discovery</a> section.</td>
</tr>
<tr>
<td></td>
<td>2. POST the RDF content describing the component (with a Content-Type header set to an RDF media type supported by the server) to the creation factory. The RDF content should be compliant with the resource shape specified for the creation factory, and that resource shape must be compatible with the resource shape for Component as described in the OSLC specification (see <a href="https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#ComponentShape">https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#ComponentShape</a>).</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>POST creationFactoryUri</td>
</tr>
<tr>
<td>PIM Services</td>
<td>OSLC PSM</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| **getProjects** | 1. Discover a query capability for components (oslc_config:Component). See OSLC Discovery section.  
2. Perform a GET on the query base, specifying which properties, if any, of the component should be returned in the query result using an `oslc.select` query parameter.  
Example: Query for components returning their name (dcterms:title) and id (dcterms:identifier)  
GET `queryBaseUri`? `oslc.select=dcterms%3Atitle%2Cdcterms%3Aidentifier` |
| **updateProject** | Execute an HTTP PUT with the revised content of the component. The RDF content should be compliant with the instance resource shape of the Component, and this should be compatible with the resource shape for Component as described in the OSLC specification (see [https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#ComponentShape](https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#ComponentShape)) |
| **deleteProject** | Not supported directly. Deletion operation is not supported for Components in the OSLC Configuration Management specification. See [https://oslc-op.github.io/oslc-specs/specs/config/config-resources.html#componentoperations](https://oslc-op.github.io/oslc-specs/specs/config/config-resources.html#componentoperations). However, providers may implement deletion operation for Components. |
| **ElementNavigation Service** | |
| **getElements** | 1. Discover a query capability for elements (an API-specific RDF type) for the specified project. See OSLC Discovery section.  
2. Perform a GET on the query base, specifying which properties, if any, of the commit should be returned in the query result using an `oslc.select` query parameter.  
Example: Query for elements returning name (dcterms:title) and identifier (dcterms:identifier).  
GET `queryBaseUri`? `oslc.select=dcterms%3Atitle%2Cdcterms%3Aidentifier` |
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
</table>
| getElementById               | In OSLC, an Element is identified by its URL. So simply perform a GET on the Element URL. To search for an Element with a given dcterms:identifier (or any other property that is deemed to return a single query result):  

1. Discover a query capability for elements (an API-specific RDF type) for the specified project. See OSLC Discovery section.  
2. Perform a GET on the query base, specifying a query for the dcterms:identifier in a oslc.where query parameter, and specifying which properties, if any, of the commit should be returned in the query result using an oslc.select query parameter.  

Example: Query for elements with identifier 123, returning all properties.  
GET queryBaseUri?
  oslc.where=dcterms%3Aidentifier%3D%22123%22&
  oslc.select=* |
| getRelationshipsByRelatedElement | 1. Discover the query capability for Elements for the particular project (and its specific commit). See OSLC Discovery section.  
2. Perform a GET on the query base, setting the oslc.where query parameter to:  

1. sysml:out = URI of the subject element to get all the outgoing Relationships from that element  
2. sysml:in = URI of the subject element to get all the incoming Relationships to the element  

Example 1: Query for Relationships outgoing from an element <http://sysml2.server.com/elements/123>:  
GET queryBaseUri?
  oslc.where=sysml:out=<http://sysml2.server.com/elements/123> and
  rdf:type=<http://omg.org/ns/sysml/v2/metamodel%23Relationship>&
  oslc.prefix=sysml=<http://omg.org/ns/sysml/v2/metamodel%23>  

GET queryBaseUri?
  oslc.where=sysml:in=<http://sysml2.server.com/elements/123> and
  rdf:type=<http://omg.org/ns/sysml/v2/metamodel%23Relationship>&
  oslc.prefix=sysml=<http://omg.org/ns/sysml/v2/metamodel%23>  

For Direction equal to both: perform and merge the separate queries on in and out values. |
<p>| getRootElements | Not Available. OSLC Query does not currently support the ability to search for resources, where certain properties (owner in this case) are not set. |</p>
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
</table>
| ProjectDataVersioning Service | 1. Discover a query capability for streams (oslc_config:Stream) for the specified project. See OSLC Discovery section. 2. Perform a GET on the query base, specifying a query for the dcterms:identifier in a oslc.where query parameter, and specifying which properties, if any, of the component should be returned in the query result using an oslc.select query parameter. Example: Query for streams with identifier 123, returning all properties. GET queryBaseUri?
   oslc.where=dcterms%3Aidentifier%3D%22123%22&
   oslc.select=* |
| getBranchById         | 1. Discover a query capability for baselines (oslc_config:Baseline) for the specified project. See OSLC Discovery section. 2. Perform a GET on the query base, specifying a query for the dcterms:identifier in a oslc.where query parameter, and specifying which properties, if any, of the component should be returned in the query result using an oslc.select query parameter. Example: Query for baselines with identifier 123, returning all properties. GET queryBaseUri?
   oslc.where=dcterms%3Aidentifier%3D%22123%22&
   oslc.select=* |
| getTagById            | Not available. The OSLC Configuration Management specification does not define any notion of a commit. Versions of specific resources may be fetched. |
| getCommits            | 1. Discover a query capability for streams (oslc_config:Stream) for the specified project. See OSLC Discovery section. 2. Perform a GET on the query base, specifying which properties, if any, of the stream should be returned in the query result using an oslc.select query parameter. Example: Query for branches returning their name (dcterms:title) and id (dcterms:identifier) GET queryBaseUri?
   oslc.select=dcterms%3Atitle%2Cdcterms%3Aidentifier |
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCommit</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a commit. New versions may be created in the context of a stream or change set. However, the OSLC Configuration Management specification does not define any means to commit a change set or deliver the changes in a stream or change set to another stream. For creating new versions of a resource in the context of a stream or a change set, see below.</td>
</tr>
<tr>
<td></td>
<td>1. Execute a PUT on version resource concept URI with configuration context parameter/header set to the URI of a stream (branch). Body contains the updated RDF representation of the element version. Note that this only supports a new commit along a branch with the previous latest commit being its previous commit. If a client wants to commit from an earlier version, they must first create a stream (branch) from that earlier baseline (commit) and then use a PUT with that new stream. The OSLC Configuration Management specification does not currently define any mechanisms for creating new versions of multiple versioned resources in a single REST operation.</td>
</tr>
<tr>
<td></td>
<td>Example 1:</td>
</tr>
<tr>
<td></td>
<td>PUT conceptResourceUri? oslc_config.context=urlEncodedStreamUri</td>
</tr>
<tr>
<td></td>
<td>Example 2:</td>
</tr>
<tr>
<td></td>
<td>Headers: Configuration-Context=streamUri</td>
</tr>
<tr>
<td></td>
<td>PUT conceptResourceUri</td>
</tr>
<tr>
<td>getCommitById</td>
<td>1. Discover a query capability for commits (a API-specific RDF type) for the specified project. See OSLC Discovery section. 2. Perform a GET on the query base, specifying a query for the dcterms:identifier in a oslc.where query parameter, and specifying which properties, if any, of the commit should be returned in the query result using an oslc.select query parameter. Example: Query for commits with identifier 123, returning all properties.</td>
</tr>
<tr>
<td></td>
<td>GET queryBaseUri? oslc.where=dcterms%3Aidentifier%3D%22123%22&amp; oslc.select=*</td>
</tr>
<tr>
<td>getCommitChange</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a commit. New versions may be created in the context of a stream or change set. However, the OSLC Configuration Management specification does not define any means to commit a change set or deliver the changes in a stream or change set to another stream.</td>
</tr>
<tr>
<td>PIM Services</td>
<td>OSLC PSM</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getCommitChangeById</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a commit. New versions may be created in the context of a stream or change set. However, the OSLC Configuration Management specification does not define any means to commit a change set or deliver the changes in a stream or change set to another stream.</td>
</tr>
<tr>
<td>deleteBranch</td>
<td>Execute an HTTP DELETE on the URI of the stream. Note that servers may reject the delete request if the stream is used or referenced by other configurations. A server may also support archiving (soft delete) a stream using a HTTP PUT with content that includes a oslc:archived &quot;true&quot;^^xsd:boolean property.</td>
</tr>
<tr>
<td>getDefaultBranch</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a default stream.</td>
</tr>
<tr>
<td>deleteTag</td>
<td>Execute an HTTP DELETE on the URI of the baseline. Note that a server might reject a request to delete a baseline if it is used or referenced by other configurations. A server may also support archiving (soft delete) a baseline using a HTTP PUT with content that includes a oslc:archived &quot;true&quot;^^xsd:boolean property.</td>
</tr>
<tr>
<td>setDefaultBranch</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a default stream.</td>
</tr>
<tr>
<td>getHeadCommit</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a commit. New versions may be created in the context of a stream or change set. However, OSLC Configuration Management does not define any means to commit a change set or deliver the changes in a stream or change set to another stream.</td>
</tr>
<tr>
<td>getTaggedCommit</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of a commit. New versions may be created in the context of a stream or change set. However, OSLC Configuration Management does not define any means to commit a change set or deliver the changes in a stream or change set to another stream. A baseline can provide a set of versioned resources but it is not a commit.</td>
</tr>
</tbody>
</table>
| getTags                      | 1. Discover a query capability for baselines (oslc_config:Baseline) for the specified project. See OSLC Discovery section.  
2. Perform a GET on the query base, specifying which properties, if any, of the baseline should be returned in the query result using an oslc.select query parameter.  
Example: Query for tags returning their name (dcterms:title) and id (dcterms:identifier)  
GET queryBaseUri?  
oslc.select=dcterms%3Atitle%2Cdcterms%3Aidentifier |
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
</table>
| createTag     | 1. Get the baselines LDPC URI from the RDF of a stream.  
2. POST the RDF representation of the baseline to the LDPC URI (with a Content-Type header set to an RDF media type supported by the server). The RDF content should be compatible with the resource shape for Baseline as described in the specification (see [https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#BaselineShape](https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#BaselineShape)) |
|               | Example: Create a baseline from a stream  
POST baselinesLdpcUri |
| createBranch  | A server might support either or both of the following:  
A) Use a creation factory:  
1. Discover a creation factory for streams (oslc_config:Stream) for the specified project. See OSLC Discovery section.  
2. POST the RDF content describing the stream (with a Content-Type header set to an RDF media type supported by the server) to the creation factory. The RDF content should be compliant with the resource shape specified for the creation factory, and that resource shape must be compatible with the resource shape for Stream as described in the specification (see [https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#StreamShape](https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#StreamShape))  
Example:  
POST creationFactoryUri  
or  
B) Use the streams LDPC of a component  
1. Get the URI of the streams LDPC from the RDF of a component.  
2. POST the RDF content describing the stream (with a Content-Type header set to an RDF media type supported by the server) to the LDPC. The RDF content should be compatible with the resource shape for Stream as described in the specification (see [https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#StreamShape](https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#StreamShape))  
Example:  
POST streamsLdpcUri |
<p>| mergeIntoBranch | Not Available |
| diffCommits    | Not Available |
| QueryService   | Not Available |</p>
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>getQueryById</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
<tr>
<td>getQueries</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
<tr>
<td>executeQueryById</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
<tr>
<td>createQuery</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
<tr>
<td>updateQuery</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
<tr>
<td>deleteQuery</td>
<td>Not Available. OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.</td>
</tr>
</tbody>
</table>
| executeQuery    | 1. Discover a query capability for the specified project. See *OSLC Discovery* section.  
                      2. Perform a GET on the query base, specifying the following. See OSLC Query Capability in OSLC Nomenclature  
                       1. The *oslc.where* query parameter to filter for the desired elements  
                       2. The *oslc.select* query parameter, to define the element properties that should be returned in the query result. |

**ExternalRelationship Service**

OSLC does/can not differentiate between properties/relationships/elements that are external or otherwise. The services below are just specific examples of the getElements and getElementById service operation mappings above.

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getExternalRelationshipsById</td>
<td>See getElementById under ElementNavigationService</td>
</tr>
<tr>
<td>getExternalRelationshipsByElementEnd</td>
<td>See getRelationshipsByRelatedElement under ElementNavigationService</td>
</tr>
<tr>
<td>getExternalRelationships</td>
<td>See getElements under Element Navigation Service</td>
</tr>
</tbody>
</table>

**ProjectUsageService**

createProjectUsage | Not available. The OSLC Configuration Management specification does not define any notion of component usage. |

deleteProjectUsage | Not available. The OSLC Configuration Management specification does not define any notion of component usage. |
<table>
<thead>
<tr>
<th>PIM Services</th>
<th>OSLC PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProjectUsages</td>
<td>Not available. The OSLC Configuration Management specification does not define any notion of component usage.</td>
</tr>
</tbody>
</table>
A Annex: Conformance Test Suite

(Normative)

A.1 ProjectService Conformance Test Cases

Operation create_project

PIM-PS-001

<table>
<thead>
<tr>
<th>Description</th>
<th>Create Project - success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. _description : String[0..1]</td>
</tr>
<tr>
<td></td>
<td>2. _name : String[1]</td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>Precondition (OCL)</td>
<td>let _record : Record = Record.allInstances()</td>
</tr>
<tr>
<td>Steps</td>
<td>Execute operation create_project(name = _name, description = _description) : Project[1]</td>
</tr>
<tr>
<td>Result</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Result, defined as project : Project[1], is a created Project</td>
</tr>
<tr>
<td></td>
<td>2. The id attribute value of the created Project is randomly generated and universally unique, including (but not limited to) not being used as the id attribute value for any previously existing Record</td>
</tr>
<tr>
<td></td>
<td>3. A Branch is created and specified as the defaultBranch attribute value for the created Project</td>
</tr>
<tr>
<td></td>
<td>4. The id attribute value of the created Branch (defaultBranch) is randomly generated and universally unique, including (but not limited to) not being used as the id attribute value for any previously existing Record.</td>
</tr>
<tr>
<td></td>
<td>5. The name attribute value of the created Branch (defaultBranch) is specified as 'main'</td>
</tr>
<tr>
<td></td>
<td>6. The description attribute value is specified as inputted</td>
</tr>
<tr>
<td></td>
<td>7. The name attribute value is specified as inputted</td>
</tr>
<tr>
<td>Postcondition (OCL)</td>
<td>let project : Project = create_project(_name, _description)</td>
</tr>
<tr>
<td></td>
<td>project-&gt;size() = 1</td>
</tr>
<tr>
<td></td>
<td>Project.allInstances()-&gt;includes(project)</td>
</tr>
<tr>
<td></td>
<td>_record.id-&gt;excludes(project.id)</td>
</tr>
<tr>
<td></td>
<td>project.defaultBranch-&gt;notEmpty()</td>
</tr>
<tr>
<td></td>
<td>_record.id-&gt;excludes(project.defaultBranch.id)</td>
</tr>
<tr>
<td></td>
<td>project.defaultBranch.name = 'main'</td>
</tr>
<tr>
<td></td>
<td>project.description = _description</td>
</tr>
<tr>
<td></td>
<td>project.name = _name</td>
</tr>
</tbody>
</table>
Operation get_projects

PIM-PS-002

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Projects - success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>None</td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>Precondition (OCL)</td>
<td></td>
</tr>
<tr>
<td>Steps</td>
<td>Execute operation get_projects() : Project[0..*]</td>
</tr>
<tr>
<td>Result</td>
<td>Result, defined as project : Project[0..*], is all of the existing Projects</td>
</tr>
<tr>
<td>Postcondition (OCL)</td>
<td>let project : Project = get_projects() project = Project.allInstances()</td>
</tr>
</tbody>
</table>

Operation get_project_by_id

PIM-PS-003

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Project by ID - exists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1. _projectId : UUID[1]</td>
</tr>
<tr>
<td>Scenario</td>
<td>A Project with an id attribute value equal to _projectId exists</td>
</tr>
<tr>
<td>Precondition (OCL)</td>
<td>let _project : Project = Project.allInstances()-&gt;select(id = _projectId) _project-&gt;size() = 1</td>
</tr>
<tr>
<td>Steps</td>
<td>Execute operation get_project_by_id(projectId = _projectId) : Project[0..1]</td>
</tr>
<tr>
<td>Result</td>
<td>Result, defined as project : Project[1], is the Project with an id attribute value equal to _projectId</td>
</tr>
<tr>
<td>Postcondition (OCL)</td>
<td>let project : Project = get_project_by_id(_projectId) project = _project</td>
</tr>
</tbody>
</table>

PIM-PS-004

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Project by ID - does not exist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1. _projectId : UUID[1]</td>
</tr>
<tr>
<td>Scenario</td>
<td>A Project with an id attribute value equal to _projectId does not exist</td>
</tr>
<tr>
<td>Precondition (OCL)</td>
<td>Project.allInstances()-&gt;select(projectId = _projectId)-&gt;isEmpty()</td>
</tr>
</tbody>
</table>
### Steps

| Execute operation | get_project_by_id(projectId = _projectId) : Project[0..1] |

### Result

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Result, defined as project : Project[0], does not include any Projects</td>
<td>2. Result communicates that a Project with the provided ID does not exist</td>
</tr>
</tbody>
</table>

### Postcondition (OCL)

```
let project : Project = get_project_by_id(_projectId)
project->isEmpty()
```

---

### A.2 ElementNavigationService Conformance Test Cases

#### Operation get_elements

**PIM-EN-001**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Elements - success</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td>2. _commit : Commit[1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Project exists</td>
</tr>
<tr>
<td>2. The inputted Commit exists</td>
</tr>
<tr>
<td>3. The inputted Commit belongs to the inputted Project</td>
</tr>
</tbody>
</table>

| Precondition (OCL) | Project.allInstances()->includes(_project) Commit.allInstances()->includes(_commit) _commit.owningProject = _project |

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute operation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result, defined as element : Element[0..*], is all Elements at the inputted Commit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Postcondition (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>let element : Element = get_elements(_project, _commit) element = _commit.version.data-&gt;select(oclIsKindOf(Element))</td>
</tr>
</tbody>
</table>

---

#### Operation get_element_by_id

**PIM-EN-002**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Element by ID - success</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td>2. _commit : Commit[1]</td>
</tr>
<tr>
<td>3. _elementId : UUID[1]</td>
</tr>
</tbody>
</table>
| Scenario | 1. The inputted Project exists  
| | 2. The inputted Commit exists  
| | 3. The inputted Commit belongs to the inputted Project  
| | 4. An Element with an id attribute value equal to _elementId exists at the inputted Commit |
| Precondition  (OCL) | Project.allInstances()->includes(_project)  
| | Commit.allInstance()->includes(_commit)  
| | _commit.owningProject = _project  
| | let _element : Element = _commit.version->select(identity.id = _elementId and data.oclIsKindOf(Element)).data  
| | _element.size() = 1 |
| Steps | Execute operation get_element_by_id(project = _project, commit = _commit, elementId = _elementId) : Element[0..1] |
| Result | Result, defined as element : Element[1], is the Element with an id attribute value equal to _elementId at the inputted Commit |
| Postcondition  (OCL) | let element : Element = get_element_by_id(_project, _commit, _elementId)  
| | element = _element |

PIM-EN-003

| Description | Get Element by ID - does not exist at Commit |
| Input | 1. _project : Project[1]  
| | 2. _commit : Commit[1]  
| | 3. _elementId : UUID[1] |
| Scenario | 1. The inputted Project exists  
| | 2. The inputted Commit exists  
| | 3. The inputted Commit belongs to the inputted Project  
| | 4. An Element with an id attribute value equal to _elementId does not exist at the inputted Commit |
| Precondition  (OCL) | Project.allInstances()->includes(_project)  
| | Commit.allInstance()->includes(_commit)  
| | _commit.owningProject = _project  
| | _commit.version->select(identity.id = _elementId and data.oclIsKindOf(Element)).data->isEmpty() |
| Steps | Execute operation get_element_by_id(project = _project, commit = _commit, elementId = _elementId) : Element[0..1] |
1. Result, defined as `element : Element[0]`, does not include any Elements
2. Result communicates that an Element with the provided ID at the inputted Commit
does not exist

### Postcondition (OCL)

```
let element : Element = get_element_by_id(_project, _commit, _id)
if element->isEmpty()
```

---

**Operation `get_relationships_by_source`**

**PIM-EN-004**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Relationships by source (Element) - success</th>
</tr>
</thead>
</table>
| **Input**                    | 1. `_project : Project[1]`
                              | 2. `_commit : Commit[1]`
                              | 3. `_elementId : UUID[1]` |
| **Scenario**                 | 1. The inputted Project exists
                              | 2. The inputted Commit exists
                              | 3. The inputted Commit belongs to the inputted Project
                              | 4. An Element with an `id` attribute value equal to `_elementId` exists at the inputted Commit |
| **Precondition (OCL)**       | Project.allInstances()\(\rightarrow\)includes(_project) 
                              | Commit.allInstances()\(\rightarrow\)includes(_commit)
                              | _commit.owningProject = _project 
                              | let _element : Element = _commit.version\(\rightarrow\)select(identity.id = _id and
                              | data.oclIsKindOf(Element)).data 
                              | _element.size() = 1 |
| **Steps**                    | Execute operation `get_relationships_by_source(project = _project, commit = _commit, elementId = _elementId) : Relationship[0..*]` |
| **Result**                   | Result, defined as `relationship : Relationship[0..*]`, is all the Relationships whose `source` attribute value includes the Element with `id` attribute value equal to `_elementId` |
| **Postcondition (OCL)**      | let relationship : Relationship = get_relationship_by_source(_project, _commit, _elementId)
                              | relationship = _commit.version\(\rightarrow\)select(data.oclIsKindOf(Relationship) and
                              | data.source\(\rightarrow\)includes(_element)) |

---

**Operation `get_relationships_by_target`**

**PIM-EN-005**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Relationships by target (Element) - success</th>
</tr>
</thead>
</table>
### A.3 ProjectDataVersioningService Conformance Test Cases

#### Operation create_branch

**PIM-PCB-001**

<table>
<thead>
<tr>
<th>Description</th>
<th>Create Branch - success</th>
</tr>
</thead>
</table>

#### Input

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>_project : Project[1]</td>
</tr>
<tr>
<td>2.</td>
<td>_head : Commit[1]</td>
</tr>
<tr>
<td>3.</td>
<td>_name : String[1]</td>
</tr>
</tbody>
</table>

#### Scenario

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The inputted Project exists</td>
</tr>
<tr>
<td>2.</td>
<td>The inputted Commit (_head) exists</td>
</tr>
<tr>
<td>3.</td>
<td>The inputted Commit (_head) belongs to the inputted Project</td>
</tr>
</tbody>
</table>
## Operation create_branch

**Precondition (OCL)**

\[
\begin{align*}
\text{Project.allInstances() -> includes(_project)} \\
\text{Commit.allInstance() -> includes(_head)} \\
\text{_head.owningProject = _project} \\
\text{let _record : Record = Record.allInstances()} \\
\end{align*}
\]

**Steps**

Execute operation \( create\_branch(project = _project, head = _head, name = _name) : \text{Branch[1]} \)

**Result**

1. Result, defined as \( \text{branch : Branch[1]} \), is a created Branch in the inputted Project
2. The \( \text{id} \) attribute value of the created Branch is randomly generated and universally unique, including (but not limited to) not being used as the \( \text{id} \) attribute value for any previously existing Record.
3. The \( \text{owningProject} \) attribute value of the created Branch is specified as inputted, i.e. equal to \( _\text{project} \)
4. The \( \text{head} \) attribute value of the created Branch is specified as inputted
5. The \( \text{name} \) attribute value of the created Branch is specified as inputted

**Postcondition (OCL)**

\[
\begin{align*}
\text{let branch : Branch = create\_branch(_project, _head, _name)} \\
\text{branch->size() = 1} \\
\text{Branch.allInstances() -> includes(branch)} \\
\text{_project.branch -> includes(branch)} \\
\text{_record.id -> excludes(branch.id)} \\
\text{branch.owningProject = _project} \\
\text{branch.head = _head} \\
\text{branch.name = _name} \\
\end{align*}
\]

## Operation get_branches

**PIM-PCB-002**

**Description**

Get Branches - success

**Input**

1. \( _\text{project : Project[1]} \)

**Scenario**

The inputted Project exists

**Precondition (OCL)**

\[
\text{Project.allInstances() -> includes(_project)}
\]

**Steps**

Execute operation \( get\_branches(project = _project) : \text{Branch[0..*]} \)

**Result**

Result, defined as \( \text{branch : Branch[0..*]} \), is all Branches in the inputted Project

**Postcondition (OCL)**

\[
\begin{align*}
\text{let branch : Branch = get\_branches(_project)} \\
\text{branch = _project.branch} \\
\end{align*}
\]
## Operation get_branch_by_id

**PIM-PCB-003**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Branch by ID - success</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td></td>
</tr>
<tr>
<td>1. _project : Project[1]</td>
<td></td>
</tr>
<tr>
<td>2. _branchId : UUID[1]</td>
<td></td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td></td>
</tr>
<tr>
<td>1. The inputted Project exists</td>
<td></td>
</tr>
<tr>
<td>2. A Branch with an id attribute value equal to _branchId exists in the inputted Project</td>
<td></td>
</tr>
<tr>
<td><strong>Precondition (OCL)</strong></td>
<td>Project.allInstances()-&gt;includes(_project)</td>
</tr>
<tr>
<td></td>
<td>let _branch : Branch = _project.branch-&gt;select(id = _branchId)</td>
</tr>
<tr>
<td></td>
<td>_branch.size() = 1</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>Execute operation get_branch_by_id(project = _project, branchId = _branchId) : Branch[0..1]</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>Result, defined as branch : Branch[1], is the Branch with an id attribute value equal to _branchId in the inputted Project</td>
</tr>
<tr>
<td><strong>Postcondition (OCL)</strong></td>
<td>let branch : Branch = get_branch_by_id(_project, _branchId)</td>
</tr>
<tr>
<td></td>
<td>branch = _branch</td>
</tr>
</tbody>
</table>

## PIM-PCB-004

<table>
<thead>
<tr>
<th>Description</th>
<th>Get Branch by ID - does not exist in Project</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td></td>
</tr>
<tr>
<td>1. _project : Project[1]</td>
<td></td>
</tr>
<tr>
<td>2. _branchId : UUID[1]</td>
<td></td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td></td>
</tr>
<tr>
<td>1. The inputted Project exists</td>
<td></td>
</tr>
<tr>
<td>2. A Branch with an id attribute value equal to _branchId does not exist in the inputted Project</td>
<td></td>
</tr>
<tr>
<td><strong>Precondition (OCL)</strong></td>
<td>Project.allInstances()-&gt;includes(_project)</td>
</tr>
<tr>
<td></td>
<td>_project.branch-&gt;select(id = _branchId)-&gt;isEmpty()</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>Execute operation get_branch_by_id(project = _project, branchId = _branchId) : Branch[0..1]</td>
</tr>
<tr>
<td>Result</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td></td>
</tr>
</tbody>
</table>
| 1. Result, defined as \( \text{branch} : \text{Branch}[0] \), does not include any Branches  
2. Result communicates that a Branch with the provided ID in the inputted Project does not exist |

**Postcondition (OCL)**

\[
\text{let branch : Branch = get\_branch\_by\_id(_project, _branchId)}  
\text{branch->isEmpty()}
\]

**Operation delete\_branch**

**PIM-PCB-005**

**Description**  
Delete Branch - success

**Input**

1. \( _\text{project} : \text{Project}[1] \)  
2. \( _\text{branchId} : \text{UUID}[1] \)

**Scenario**

1. The inputted Project exists  
2. A Branch with an \( \text{id} \) attribute value equal to \( _\text{branchId} \) exists in the inputted Project

**Precondition (OCL)**

\[
\text{Project.allInstances()]->includes(_project)  
\text{let _branch : Branch = _project.branch->select(id = _branchId)}  
\text{_branch.size() = 1}
\]

**Steps**

Execute operation \( \text{delete\_branch(project = _project, branchId = _branchId)} : \text{Branch}[0..1] \)

**Result**

1. Result, defined as \( \text{branch} : \text{Branch}[1] \), is the Branch with an \( \text{id} \) attribute value equal to \( _\text{branchId} \) in the inputted Project  
2. Branch with an \( \text{id} \) attribute value equal to \( _\text{branchId} \) does not exist

**Postcondition (OCL)**

\[
\text{let branch : Branch = delete\_branch(_project, _branchId)}  
\text{_project.branch->excludes(branch)}  
\text{Branch.allInstances()]->excludes(branch)}  
\text{_project.branch->select(id = _branchId)->isEmpty()  
\text{Branch.allInstances()]->select(id = _branchId)->isEmpty()}
\]

**PIM-PCB-006**

**Description**  
Delete Branch - does not exist in Project

**Input**

1. \( _\text{project} : \text{Project}[1] \)  
2. \( _\text{branchId} : \text{UUID}[1] \)
### Scenario

1. The inputted Project exists
2. A Branch with an `id` attribute value equal to `_branchId` does not exist in the inputted Project

### Precondition (OCL)

```
Project.allInstances()->includes(_project)
_project.branch->select(id = _branchId)->isEmpty()
```

### Steps

Execute operation `delete_branch(project = _project, branchId = _branchId) : Branch[0..1]`

### Result

1. Result, defined as `branch : Branch[0]`, does not include any Branches
2. Result communicates that a Branch with the provided ID in the inputted Project does not exist

### Postcondition (OCL)

```
let branch : Branch = delete_branch(_project, _branchId)
branch->isEmpty()
```

---

**Operation get_default_branch**

**PIM-PCB-007**

### Description

Get default Branch - success

### Input

1. `_project : Project[1]`

### Scenario

The inputted Project exists

### Precondition (OCL)

```
Project.allInstances()->includes(_project)
```

### Steps

Execute operation `get_default_branch(project = _project) : Branch[1]`

### Result

1. Result, defined as `branch : Branch[1]`, is the `defaultBranch` attribute value of the inputted Project

### Postcondition (OCL)

```
let branch = get_default_branch(_project)
branch = _project.defaultBranch
```

---

**Operation set_default_branch**

**PIM-PCB-008**

### Description

Set default Branch - success
### PIM-PCB-009

**Description**
Set default Branch - does not exist in Project

| Input | 1. _project : Project[1]  
2. _branchId : UUID[1] |

| Scenario | 1. The inputted Project exists  
2. A Branch with an id attribute value equal to _branchId does not exist in the inputted Project |

| Precondition (OCL) | Project.allInstances()->includes(_project)  
_project.branch->select(id = _branchId)->isEmpty()  
let _defaultBranch : Branch = _project.defaultBranch |

| Steps | Execute operation set_default_branch(project = _project, branchId = _branchId) : Project[1] |

| Result | 1. Result, defined as project : Project[1], is the inputted Project  
2. The defaultBranch attribute value of the inputted Project is specified as the Branch with an id attribute value equal to _branchId |

| Postcondition (OCL) | let project : Project = set_default_branch(_project, _branchId)  
project.defaultBranch = _defaultBranch |
Result

1. Result, defined as `result : Project[1]`, is the inputted Project
2. The `defaultBranch` attribute value of the inputted Project is unchanged
3. Result communicates that a Branch with the provided ID in the inputted Project does not exist

Postcondition

(OCL)

| let project : Project = set_default_branch(_project, _branchId) |
| project.defaultBranch = _defaultBranch |

Operation create_commit

PIM-PCB-010

Description

Create Commit - success

Input

1. `_project : Project[1]`
2. `_change : DataVersion[1..*]`
3. `_branch : Branch[0..1]`
4. `_previousCommit : Commit[0..*]`

Scenario

1. The inputted Project exists
2. The inputted Branch, optional and if provided, exists in the inputted Project
3. The inputted Commits (`_previousCommit`), optional and if provided, exists in the inputted Commit

Precondition

(OCL)

| Project.allInstances() -> includes(_project) |
| _branch -> isEmpty() or _project.branch -> includes(_branch) |
| _previousCommit -> isEmpty() or _previousCommit -> forAll(owningProject = _project) |

| let _record : Record = Record.allInstances() |
| let _head : Commit = _branch.head |

Steps

Execute operation `create_commit(project = _project, change = _change, branch = _branch, previousCommit = _previousCommit) : Commit[1]`
### Result

1. Result, defined as `commit : Commit[1]`, is a created Commit
2. The `id` attribute value of the created Commit is randomly generated and universally unique, including (but not limited to) not being used as the `id` attribute value for any previously existing Record.
3. The `owningProject` attribute value of the created Commit is specified as inputted `_project`
4. The `change` attribute value of the created Commit is specified as inputted
5. The `previousCommit` attribute value of the created Commit is specified as the union of the head attribute value of the inputted Branch, if provided, and the inputted Commits `_previousCommit`
6. The `version` attribute value of the created Commit includes the new changes
7. The `head` attribute value of the inputted Branch, if provided, is updated to the created Commit

### Postcondition (OCL)

```ocl
define Postcondition (OCL)
let commit : Commit = create_commit(_project, _change, _branch, _previousCommit)
commit->size() = 1
_record.id->excludes(commit.id)
commit.owningProject = _project
commit.change = _change
commit.previousCommit = _head->union(_previousCommit)
commit.version->includes(_change)
_branch->isEmpty() or _branch.head = commit
```  

### Operation get_commit_by_id

**PIM-PCB-011**

**Description**
Get commit by ID - success

**Input**
1. `_project : Project[1]`
2. `_commitId : UUID[1]`

**Scenario**
1. The inputted Project exists
2. A Commit with an `id` attribute value equal to `_commitId` exists in the inputted Project

**Precondition (OCL)**

```ocl
let _commit : Commit = _project.commit->select(id = _commitId)
_commit.size() = 1
```  

**Steps**
Execute operation `get_commit_by_id(project = _project, commitId = _commitId) : Commit[0..1]`

**Result**
Result, defined as `commit : Commit[1]`, is the Commit with an `id` attribute value equal to `_commitId` in the inputted Project
### Postcondition (OCL)

\[
\text{let commit} : \text{Commit} = \text{get-commit-by-id}(_\text{project}, _\text{commitId}) \\
\text{commit} = _\text{commit}
\]

### PIM-PCB-012

**Description**
Get commit by ID - does not exist in Project

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td>2. _commitId : UUID[1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Project exists</td>
</tr>
<tr>
<td>2. A Commit with an id attribute value equal to _commitId does not exist in the inputted Project</td>
</tr>
</tbody>
</table>

**Precondition (OCL)**

\[
\text{Project.allInstances() -> includes(_project)} \\
_\text{project.commit} -> \text{select(id = _commitId) -> isEmpty()}
\]

**Steps**

Execute operation get_commit_by_id(project = _project, commitId = _commitId) : Commit[0..1]

**Result**
Result, defined as commit : Commit[0], does not include any Commits

**Postcondition (OCL)**

\[
\text{let commit} : \text{Commit} = \text{get-commit-by-id}(_\text{project}, _\text{commitId}) \\
\text{commit} -> \text{isEmpty()}
\]

---

### Operation get_head

**PIM-PCB-013**

**Description**
Get head Commit of Branch - success

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td>2. _branch : Branch[1]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Project exists</td>
</tr>
<tr>
<td>2. The inputted Branch exists</td>
</tr>
<tr>
<td>3. The inputted Branch belongs to the inputted Project</td>
</tr>
</tbody>
</table>

**Precondition (OCL)**

\[
\text{Project.allInstances() -> includes(_project)} \\
\text{Branch.allInstances() -> includes(_branch)} \\
_\text{branch.owningProject} = _\text{project}
\]

**Steps**

Execute operation get_head(project = _project, branch = _branch) : Commit[0..1]
<table>
<thead>
<tr>
<th><strong>Result</strong></th>
<th>Result, defined as ( \text{head} : \text{Commit}[0..1] ), is the Commit that is the head attribute value of the inputted Branch</th>
</tr>
</thead>
</table>
| **Postcondition (OCL)** | \[
\begin{align*}
\text{let head} & : \text{Commit} = \text{get\_head}(_\text{project}, _\text{branch}) \\
\text{Bag}[0, 1]\rightarrow\text{includes(head}\rightarrow\text{size()}) \\
\text{head} & = _\text{project}.\text{head}
\end{align*}
\] |

**PIM-PCB-014**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Get head Commit of Branch - Branch does not exist</th>
</tr>
</thead>
</table>
| **Input** | 1. \(_\text{project} : \text{Project}[1]\) \\
2. \(_\text{branch} : \text{Branch}[1]\) |
| **Scenario** | 1. The inputted Project exists \\
2. The inputted Branch does not exist |
| **Precondition (OCL)** | \[
\text{Project.allInstances()}\rightarrow\text{includes(_project)} \\
\text{Branch.allInstances()}\rightarrow\text{excludes(_branch)}
\] |
| **Steps** | Execute operation \( \text{get\_head(project = _\text{project}, branch = _\text{branch})} : \text{Commit}[0..1] \) |
| **Result** | 1. Result, defined as \( \text{head} : \text{Commit}[0] \), does not include any Commits \\
2. Result communicates that the inputted Branch does not exist |
| **Postcondition (OCL)** | \[
\begin{align*}
\text{let head} & : \text{Commit} = \text{get\_head}(_\text{project}, _\text{branch}) \\
\text{head}\rightarrow\text{isEmpty()}
\end{align*}
\] |

**PIM-PCB-015**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Get head Commit of Branch - Branch not in Project</th>
</tr>
</thead>
</table>
| **Input** | 1. \(_\text{project} : \text{Project}[1]\) \\
2. \(_\text{branch} : \text{Branch}[1]\) |
| **Scenario** | 1. The inputted Project exists \\
2. The inputted Branch exists \\
3. The inputted Branch does not belong to the inputted Project |
Precondition (OCL)
Project.allInstances() \rightarrow includes(_project)
Branch.allInstance() \rightarrow includes(_branch)
_branch.owningProject <> _project

Steps
Execute operation get_head(project = _project, branch = _branch) : Commit[0..1]

Result
1. Result, defined as head : Commit[0], does not include any Commits
2. Result communicates that the provided input is invalid

Postcondition (OCL)
let head : Commit = get_head(_project, _branch)
head->isEmpty()

A.4 QueryService Conformance Test Cases

Operation create_query
PIM-QS-001

<table>
<thead>
<tr>
<th>Description</th>
<th>Create Query - success</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td>2. _select : String[0..*]</td>
</tr>
<tr>
<td>3. _scope : DataIdentity[0..*]</td>
</tr>
<tr>
<td>4. _where : Constraint[0..1]</td>
</tr>
<tr>
<td>5. _orderBy : String[0..*]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Project exists</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project.allInstances() \rightarrow includes(_project)</td>
</tr>
<tr>
<td>let _record : Record = Record.allInstances()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute operation create_query(project = _project, select = _select, scope = _scope, where = _where, orderBy = _orderBy) : Query[1]</td>
</tr>
</tbody>
</table>
### Result
1. Result, defined as \( \text{query} : \text{Query}[1] \), is a created Query in the inputted Project
2. The \text{id} attribute value of the created Query is randomly generated and universally unique, including (but not limited to) not being used as the \text{id} attribute value for any previously existing Record.
3. The \text{owningProject} attribute value of the created Query is specified as inputted, i.e. equal to \(_\text{project}\)
4. The \text{select} attribute value of the created Query is specified as inputted
5. The \text{scope} attribute value of the created Query is specified as inputted
6. The \text{where} attribute value of the created Query is specified as inputted
7. The \text{orderBy} attribute value of the created Query is specified as inputted

### Postcondition (OCL)
\[
\begin{align*}
\text{let } \text{query} : \text{Query} &= \text{create}_\text{query}(\_\text{project}, \_\text{select}, \_\text{scope}, \_\text{where}, \\
& \quad \_\text{orderBy}) \\
\text{query->size()} &= 1 \\
\text{Query.allInstances() includes(query)} \\
\_\text{project.query includes(query)} \\
\_\text{record.id excludes(query.id)} \\
\text{query.owningProject = _project} \\
\text{query.select = _select} \\
\text{query.scope = _scope} \\
\text{query.where = _where} \\
\text{query.orderBy = orderBy}
\end{align*}
\]

### PIM-QS-002

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute Query - success</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. _query : \text{Query}[1]</td>
<td></td>
</tr>
<tr>
<td>2. _commit : \text{Commit}[1]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Query exists</td>
<td></td>
</tr>
<tr>
<td>2. The inputted Commit exists</td>
<td></td>
</tr>
<tr>
<td>3. The inputted Query and Commit both belong to the same Project</td>
<td></td>
</tr>
</tbody>
</table>

### Precondition (OCL)
\[
\begin{align*}
\text{Query.allInstances() includes(_query)} \\
\text{Commit.allInstances() includes(_commit)} \\
\_\text{query.owningProject = _commit.owningProject}
\end{align*}
\]

| Steps                | Execute operation \text{execute}_\text{query}(query = _\text{query}, commit = _\text{commit}) : \text{Data}[0..*] |

<table>
<thead>
<tr>
<th>Result</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Result, defined as \text{data} : \text{Data}[0..*], \text{is all of the Datas at the inputted Commit that satisfy the conditions of the inputted Query}</td>
<td></td>
</tr>
<tr>
<td>2. If inputted Query has a non-empty \text{scope} attribute value, result only includes Data within the Query's scope</td>
<td></td>
</tr>
</tbody>
</table>
A.5 ExternalRelationshipService Test Cases

Operation get_external_relationships

PIM-ER-001

<table>
<thead>
<tr>
<th>Description</th>
<th>Get ExternalRelationships - success</th>
</tr>
</thead>
</table>

### Input

1. `_project` : `Project[1]`
2. `_commit` : `Commit[1]`

### Scenario

1. The inputted Project exists
2. The inputted Commit exists
3. The inputted Commit belongs to the inputted Project

### Precondition (OCL)

```ocl
Project.allInstances()->includes(_project)
Commit.allInstance()->includes(_commit)
_commit.owningProject = _project
```

### Steps

Execute operation `get_external_relationships(project = _project, commit = _commit) : ExternalRelationship[0..*]`

### Result

1. Result, defined as `externalRelationship : ExternalRelationship[0..*]`, is all of the existing ExternalRelationships at the inputted Commit

### Postcondition (OCL)

```ocl
let externalRelationship : ExternalRelationship =
get_external_relationships(_project, _commit)
externalRelationship =
_commit.version.data->select(oclIsKindOf(ExternalRelationship))
```

Operation get_external_relationship_by_id

PIM-ER-002

<table>
<thead>
<tr>
<th>Description</th>
<th>Get ExternalRelationship by ID - success</th>
</tr>
</thead>
</table>

### Input

1. `_project` : `Project[1]`
2. `_commit` : `Commit[1]`
3. `_externalRelationshipId` : `UUID[1]`
### Scenario

1. The inputted Project exists
2. The inputted Commit exists
3. The inputted Commit belongs to the inputted Project
4. An ExternalRelationship with an id attribute value equal to `_externalRelationshipId` exists at the inputted Commit

### Precondition (OCL)

```plaintext
Project.allInstances()->includes(_project)
Commit.allInstance()->includes(_commit)
_commit.owningProject = _project
let _externalRelationship : ExternalRelationship =
  _commit.version->select(identity.id = _externalRelationshipId and
data.oclIsTypeOf(ExternalRelationship)).data
 _externalRelationship.size() = 1
```

### Steps

Execute operation `get_external_relationship_by_id(project = _project,
commit = _commit,
externalRelationshipId = _externalRelationshipId) : ExternalRelationship[0..1]`

### Result

1. Result, defined as `externalRelationship : ExternalRelationship[1]`, is the Element with an id attribute value equal to `_externalRelationshipId` at the inputted Commit

### Postcondition (OCL)

```plaintext
let externalRelationship : ExternalRelationship =
get_external_relationship_by_id(_project, _commit,
 _externalRelationshipId)
externalRelationship = _externalRelationship
```

## PIM-ER-003

### Description
Get ExternalRelationship by ID - does not exist at Commit

### Input

1. `_project` : Project[1]
2. `_commit` : Commit[1]
3. `_externalRelationshipId` : UUID[1]

### Scenario

1. The inputted Project exists
2. The inputted Commit exists
3. The inputted Commit belongs to the inputted Project
4. An ExternalRelationship with an id attribute value equal to `_externalRelationshipId` does not exist at the inputted Commit
### Precondition (OCL)

| Project.allInstances()->includes(_project) |
| Commit.allInstance()->includes(_commit) |
| _commit.owningProject = _project |
| _commit.version->select(identity.id = _externalRelationshipId and data.oclIsTypeOf(ExternalRelationship)).data->isEmpty() |

### Steps

**Execute operation**

get_external_relationship_by_id(project = _project, commit = _commit, externalRelationshipId = _externalRelationshipId) : ExternalRelationship[0..1]

### Result

1. Result, defined as externalRelationship : ExternalRelationship[0], does not include any ExternalRelationships
2. Result communicates that an ExternalRelationship with the provided ID at the inputted Commit does not exist

### Postcondition (OCL)

let externalRelationship : ExternalRelationship = get_external_relationship_by_id(_project, _commit, _externalRelationshipId) 

externalRelationship->isEmpty()

---

**A.6 ProjectUsageService Conformance Test Cases**

**Operation get_project_usages**

**PIM-PU-001**

<table>
<thead>
<tr>
<th>Description</th>
<th>Get ExternalRelationships - success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>1. _project : Project[1]</td>
<td></td>
</tr>
<tr>
<td>2. _commit : Commit[1]</td>
<td></td>
</tr>
<tr>
<td>Scenario</td>
<td></td>
</tr>
<tr>
<td>1. The inputted Project exists</td>
<td></td>
</tr>
<tr>
<td>2. The inputted Commit exists</td>
<td></td>
</tr>
<tr>
<td>3. The inputted Commit belongs to the inputted Project</td>
<td></td>
</tr>
<tr>
<td>Precondition (OCL)</td>
<td>Project.allInstances()-&gt;includes(_project)</td>
</tr>
<tr>
<td>Commit.allInstance()-&gt;includes(_commit)</td>
<td></td>
</tr>
<tr>
<td>_commit.owningProject = _project</td>
<td></td>
</tr>
<tr>
<td>Steps</td>
<td>Execute operation get_project_usages(project = _project, commit = _commit) : ProjectUsage[0..*]</td>
</tr>
</tbody>
</table>
### Result

1. Result, defined as `projectUsage : ProjectUsage[0..*]`, is all of the existing ProjectUsages at the inputted Commit

### Postcondition (OCL)

```plaintext
let projectUsage : ProjectUsage = 
get_project_usages(_project, _commit)
projectUsage = 
_commit.version.data->select(oclIsKindOf(ProjectUsage))
```

### A.7 Cross-Cutting Conformance Test Cases

#### Operations with Invalid Input

**PIM-CC-001**

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute operation - missing Project input</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>1. <code>_project : Project[0]</code></td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Precondition (OCL)</strong></td>
<td>Execute any of the following operations, defined as <code>x(project : Project[1], ...)</code>:</td>
</tr>
</tbody>
</table>
| **Steps**   | • `get_elements(project = _project, ...)`
|            | • `get_element_by_id(project = _project, ...)`
|            | • `get_relationships_by_source(project = _project, ...)`
|            | • `get_relationships_by_target(project = _project, ...)`
|            | • `create_branch(project = _project, ...)`
|            | • `get_branches(project = _project)`
|            | • `get_branch_by_id(project = _project, ...)`
|            | • `delete_branch(project = _project, ...)`
|            | • `get_default_branch(project = _project)`
|            | • `set_default_branch(project = _project, ...)`
|            | • `create_commit(project = _project, ...)`
|            | • `get_commit_by_id(project = _project, ...)`
|            | • `get_head(project = _project, ...)`
|            | • `create_query(project = _project, ...)`
| **Result**  | 1. Result, defined as `result : OclAny[0]`, does not include any OclAny
|            | 2. Result communicates that `project` is a required input |
| **Postcondition (OCL)** | `let result : OclAny = x(_project, ...) result->isEmpty()` |
### PIM-CC-002

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute operation - missing Commit input</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>1. _commit : Commit[0]</td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>Execute any of the following operations, defined as (x(\text{commit} : \text{Commit}[1], \ldots)):</td>
</tr>
<tr>
<td><strong>Steps</strong></td>
<td>• get_elements(..., commit = _commit)</td>
</tr>
<tr>
<td></td>
<td>• get_element_by_id(..., commit = _commit, ...)</td>
</tr>
<tr>
<td></td>
<td>• get_relationships_by_source(..., commit = _commit, ...)</td>
</tr>
<tr>
<td></td>
<td>• get_relationships_by_target(..., commit = _commit, ...)</td>
</tr>
<tr>
<td></td>
<td>• create_branch(..., head = _commit, ...)</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>1. Result, defined as result : OclAny[0], does not include any OclAny</td>
</tr>
<tr>
<td></td>
<td>2. Result communicates that commit is a required input</td>
</tr>
<tr>
<td><strong>Postcondition</strong></td>
<td>let result : OclAny = x(_commit, ...)</td>
</tr>
<tr>
<td></td>
<td>result-&gt;isEmpty()</td>
</tr>
</tbody>
</table>

### PIM-CC-003

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute operation - Project input does not exist</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>1. _project : Project[1]</td>
</tr>
<tr>
<td><strong>Scenario</strong></td>
<td>The inputted Project does not exist</td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
<td>Project.allInstances()-&gt;excludes(_project)</td>
</tr>
</tbody>
</table>
| Steps | Execute any of the following operations, defined as $x$ (project : Project[1], ...):
| | $\cdot$ get_elements (project = _project, ...)
| | $\cdot$ get_element_by_id (project = _project, ...)
| | $\cdot$ get_relationships_by_source (project = _project, ...)
| | $\cdot$ get_relationships_by_target (project = _project, ...)
| | $\cdot$ create_branch (project = _project, ...)
| | $\cdot$ get_branches (project = _project)
| | $\cdot$ get_branch_by_id (project = _project, ...)
| | $\cdot$ delete_branch (project = _project, ...)
| | $\cdot$ get_default_branch (project = _project)
| | $\cdot$ set_default_branch (project = _project)
| | $\cdot$ create_commit (project = _project, ...)
| | $\cdot$ get_commit_by_id (project = _project, ...)
| | $\cdot$ get_head (project = _project, ...)
| | $\cdot$ create_query (project = _project, ...)

| Result | 1. Result, defined as result : OclAny[0], does not include any OclAny
| | 2. Result communicates that the inputted Project does not exist

| Postcondition (OCL) | let result : OclAny = x(_project, ...)
| | result->isEmpty()

| PIM-CC-004 |
| Description | Execute operation - Commit input does not exist |
| Input | 1. _commit : Commit[1] |
| Scenario | The inputted Commit does not exist |
| Precondition (OCL) | Commit.allInstances()->excludes(_commit) |

| Steps | Execute any of the following operations, defined as $x$ (commit : Commit[1], ...):
| | $\cdot$ get_elements (..., commit = _commit)
| | $\cdot$ get_element_by_id (..., commit = _commit, ...)
| | $\cdot$ get_relationships_by_source (..., commit = _commit, ...)
| | $\cdot$ get_relationships_by_target (..., commit = _commit, ...)
| | $\cdot$ create_branch (..., head = _commit, ...)

| Result | 1. Result, defined as result : OclAny[0], does not include any OclAny
| | 2. Result communicates that the inputted Commit does not exist |
### PIM-CC-005

**Description**
Execute operation - Commit input is not owned by Project input

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <code>_project : Project[1]</code></td>
</tr>
<tr>
<td>2. <code>_commit : Commit[1]</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The inputted Project exists</td>
</tr>
<tr>
<td>2. The inputted Commit does not exist</td>
</tr>
<tr>
<td>3. The inputted Commit does not belong to the inputted Project</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Precondition (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project.allInstances()-&gt;includes(_project)</td>
</tr>
<tr>
<td>Commit.allInstances()-&gt;includes(_commit)</td>
</tr>
<tr>
<td>_commit.owningProject &lt;&gt; _project</td>
</tr>
</tbody>
</table>

**Steps**
Execute any of the following operations, defined as `x(project : Project[1], commit : Commit[1], ...)`:  

- `get_elements(project = _project, commit = _commit)`  
- `get_element_by_id(project = _project, commit = _commit, ...)`  
- `get_relationships_by_source(project = _project, commit = _commit, ...)`  
- `get_relationships_by_target(project = _project, commit = _commit, ...)`  
- `create_branch(project = _project, head = _commit, ...)`

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Result, defined as <code>result : OclAny[0]</code>, does not include any OclAny</td>
</tr>
<tr>
<td>2. Result communicates that the provided input is invalid</td>
</tr>
</tbody>
</table>

**Postcondition (OCL)**

```plaintext
let result : OclAny = x(_commit, ...)
result->isEmpty()
```

### PIM-CC-006

**Description**
Execute operation - missing name input

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <code>_name : String[0]</code></td>
</tr>
</tbody>
</table>

**Result**

1. Result, defined as `result : OclAny[0]`, does not include any OclAny
2. Result communicates that the provided input is invalid

**Postcondition (OCL)**

```plaintext
let result : OclAny = x(_project, _commit, ...)
result->isEmpty()
```
<table>
<thead>
<tr>
<th>Precondition (OCL)</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute any of the following operations, defined as (x(\text{name} : \text{String}[1], ...) : \text{OclAny}:)</td>
<td></td>
</tr>
<tr>
<td>• (\text{create_project}(\text{name} = _\text{name}, ...))</td>
<td></td>
</tr>
<tr>
<td>• (\text{create_branch}(..., \text{name} = _\text{name}, ...))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Result, defined as (\text{result} : \text{OclAny}[0]), does not include any (\text{OclAny})</td>
</tr>
<tr>
<td>2. Result communicates that (\text{name}) is a required input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Postcondition (OCL)</th>
</tr>
</thead>
</table>
| \(\text{let result} : \text{OclAny} = x(_\text{name}, ...)
\text{result}->\text{isEmpty}()\) |

PIM-CC-007

<table>
<thead>
<tr>
<th>Description</th>
<th>Execute operation - missing UUID input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(1. _\text{id} : \text{UUID}[0])</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scenario</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Precondition (OCL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute any of the following operations, defined as (x(\text{id} : \text{UUID}[1], ...) : \text{OclAny}:)</td>
</tr>
<tr>
<td>• (\text{get_project_by_id}(\text{projectId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{get_element_by_id}(..., \text{elementId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{get_relationships_by_source}(..., \text{elementId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{get_relationships_by_target}(..., \text{elementId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{get_branch_by_id}(..., \text{branchId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{delete_branch}(..., \text{branchId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{set_default_branch}(..., \text{branchId} = _\text{id}))</td>
</tr>
<tr>
<td>• (\text{get_commit_by_id}(..., \text{commitId} = _\text{id}))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Result, defined as (\text{result} : \text{OclAny}[0]), does not include any (\text{OclAny})</td>
</tr>
<tr>
<td>2. Result communicates that the input for which (_\text{id}) is used is a required input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Postcondition (OCL)</th>
</tr>
</thead>
</table>
| \(\text{let result} : \text{OclAny} = x(_\text{id}, ...)
\text{result}->\text{isEmpty}()\) |
B Annex: API and Services Examples and Cookbook

(Informative)

B.1 Examples

In this section, examples of REST/HTTP PSM (API) requests and responses are presented. The examples are organized per the tags in the OpenAPI specification that group individual endpoints, such as Project, Branch, Commit, and Element. The tags defined in the OpenAPI specification of the REST/HTTP PSM are generally organized by PIM Services. For each request, the curl notation and the resulting response with JSON data are presented.

The term Protocol://FQDN in the request URLs indicates the protocol (e.g. http/https) and the Fully Qualified Domain Name (FQDN) of the System Modeling API and Services Provider, e.g. https://sysml2-test.omg.org.

Project

GET /projects

Gets all the projects.

Request:

curl -X GET "Protocol://FQDN/projects" -H "accept: application/json"

Response:

```json
[
  {
    "@id": "09257c67-2dce-4177-a95a-f2ede95b3ef3",
    "@type": "Project",
    "defaultBranch": {
      "@id": "71904fd1-2d86-4bb3-a626-ca94097f746c"
    },
    "description": null,
    "name": "1c-Parts Tree Redefinition Mon Oct 18 21:43:35 CDT 2021"
  },
  {
    "@id": "0d885964-c176-4006-9809-4c74621da0f0",
    "@type": "Project",
    "defaultBranch": {
      "@id": "0a021fc1-b70c-4256-9342-41ed383d3933"
    },
    "description": null,
    "name": "Hello SysML v2"
  },
  {
    "@id": "11507ece-f27b-4b01-bc7e-1fe8cb4b665c",
    "@type": "Project",
    "defaultBranch": {
      "@id": "3e01a5f9-95bc-40de-8332-3d0d2c111d55"
    },
    "description": null,
```
GET /projects/{projectld}

Gets the project with id {projectld}.

Request:

curl -X GET "protocol://FQDN/projects/0d885964-c176-4006-9809-4c74621da0f0" -H "accept: application/json"

Response:

{
  "@id": "0d885964-c176-4006-9809-4c74621da0f0",
  "@type": "Project",
  "defaultBranch": {
    "@id": "0a021fcl-b70c-4256-9342-41ed383d3933",
    "description": null,
    "name": "Hello SysML v2"
  }
}

POST /projects

Creates a new project using the project data specified in the body of the request.

Request:

  "@type": "Project",
  "description": "Getting started with SysML v2 REST/HTTP API",
  "name": "SysML v2 API Start"
}"

Response (body):

{
  "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288",
  "@type": "Project",
  "defaultBranch": {
    "@id": "e9e33b09-ad9b-45ea-a85b-107dc862clff",
    "description": "Getting started with SysML v2 REST/HTTP API",
    "name": "SysML v2 API Start"
  }
}

Branch

GET /projects/{projectld}/branches

Gets all the branches in the project with id {projectld}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/branches" -H "accept: application/json"
Response (body):

[  
  {  
    "@id": "e9e33b09-ad9b-45ea-a85b-107dc862c1ff",  
    "@type": "Branch",  
    "head": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    },  
    "name": "main",  
    "owningProject": {  
      "@id": "6b5e80d1-7291-4ef-880c-c462ca5a3288"  
    },  
    "referencedCommit": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    },  
    "timestamp": "2021-11-06T15:45:34.967737-04:00"  
  },  
  {  
    "@id": "feefb263-f330-49da-b679-1d65602bf611",  
    "@type": "Branch",  
    "head": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    },  
    "name": null,  
    "owningProject": {  
      "@id": "6b5e80d1-7291-4ef-880c-c462ca5a3288"  
    },  
    "referencedCommit": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    },  
    "timestamp": "2021-11-06T15:55:57.332349-04:00"  
  }
]

GET /projects/{projectId}/branches/{branchId}

Gets the branch with id {branchId} in the project with id {projectId}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4ef-880c-c462ca5a3288/branches/e9e33b09-ad9b-45ea-a85b-107dc862c1ff" -H "accept: application/json"

Response (body):

[  
  {  
    "@id": "e9e33b09-ad9b-45ea-a85b-107dc862c1ff",  
    "@type": "Branch",  
    "head": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    },  
    "name": "main",  
    "owningProject": {  
      "@id": "6b5e80d1-7291-4ef-880c-c462ca5a3288"  
    },  
    "referencedCommit": {  
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"  
    }  
  }
]
POST /projects/{projectId}/branches

Creates a new branch in the project with id {projectId} using the branch data specified in the body of the request.

Request:
curl -X POST "Protocol://FQDN/projects/6b5e80d1-7291-4eeef-880c-c462ca5a3288/branches" -H "accept: application/json" -H "Content-Type: application/json" -d "{ "@type": "Branch", "head": { "@id": "068440fd-9294-4acf-bc61-00b9156c9dde" }, "name": "SysML v2 API Explore Branch", "owningProject": { "@id": "6b5e80d1-7291-4eeef-880c-c462ca5a3288" }}"

Response (body):

{ "@id": "feefb263-f330-49da-b679-1d65602bf611", "@type": "Branch", "head": { "@id": "068440fd-9294-4acf-bc61-00b9156c9dde" }, "name": null, "owningProject": { "@id": "6b5e80d1-7291-4eeef-880c-c462ca5a3288" }, "referencedCommit": { "@id": "068440fd-9294-4acf-bc61-00b9156c9dde" }, "timestamp": "2021-11-06T15:55:57.332349-04:00" }

Tag

GET /projects/{projectId}/tags

Gets all the tags in the project with id {projectId}.

Request:
curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eeef-880c-c462ca5a3288/tags" -H "accept: application/json"

Response:

[ { "@id": "d4657183-c4c1-45e9-90ac-dc984fc7c488", "@type": "Tag", "name": "Release 1", "owningProject": { "@id": "6b5e80d1-7291-4eeef-880c-c462ca5a3288" }, "referencedCommit": { "@id": "068440fd-9294-4acf-bc61-00b9156c9dde" }, "taggedCommit": { } } ]
POST /projects/{projectId}/tags

Creates a new tag in the project with id {projectId} using the tag data specified in the body of the request.

Request:

curl -X POST "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/tags" -H "accept: application/json" -H "Content-Type: application/json" -d "{"name":"Release 1","taggedCommit": {"@id": "068440fd-9294-4acf-bc61-00b9156c9dde"}}"

Response (body):

{
   "@id": "d4657183-c4c1-45e9-90ac-dc984fc7c488",
   "@type": "Tag",
   "name": "Release 1",
   "owningProject": {
      "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
   },
   "referencedCommit": {
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
   },
   "taggedCommit": {
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
   },
   "timestamp": "2021-11-06T16:18:42.992149-04:00"
}
Commit

GET /projects/{projectId}/commits

Gets all the commits in a project with id {projectId}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/commits" -H "accept: application/json"

Response:

[
  {
    "@id": "068440fd-9294-4acf-bc61-00b9156c9dde",
    "@type": "Commit",
    "owningProject": {
      "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
    },
    "previousCommit": null,
    "timestamp": "2021-11-06T15:54:11.867499-04:00"
  },
  {
    "@id": "cd4e9929-a51f-4f68-b87e-42b63083d9e6",
    "@type": "Commit",
    "owningProject": {
      "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
    },
    "previousCommit": {
      "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
    },
    "timestamp": "2021-11-06T16:39:46.579269-04:00"
  }
]

GET /projects/{projectId}/commits/{commitId}

Gets the commit with id {commitId} in the project with id {projectId}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/commits/cd4e9929-a51f-4f68-b87e-42b63083d9e6" -H "accept: application/json"

Response (body):

{
  "@id": "cd4e9929-a51f-4f68-b87e-42b63083d9e6",
  "@type": "Commit",
  "change": [
    {
      "@id": "635e54a6-f091-4925-b784-96c5243ca99d",
      "identity": {}
POST /projects/{projectId}/commits

Creates a new commit in the given project with id {projectId} using the commit data specified in the body of the request. If the branch is not specified ({branchId} query parameter is absent), the default branch of the project will reference the new commit.

Request:

curl -X POST "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/commits" 
-H "accept: application/json" -H "Content-Type: application/json" -d "{ "@type": "Commit", 
"change": [ { 
"@type": "DataVersion", 
"payload": { 
"@type": "PartDefinition", 
"name": "Vehicle_B" 
}, 
"identity": { 
"@type": "DataIdentity" 
} 
}, 
"containingProject": { 
"id": "6b5e80d1-7291-4eef-880c-c462ca5a3288" 
}, 
"previousCommit": { 
"id": "d3b258d4-c2e3-4385-b044-e32f535ae673" 
} 
}"}

Response:

{ 
"@id": "42122503-70f1-42e9-ac6d-9b1ea2790138", 
"@type": "Commit", 
"owningProject": { 
"@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288" 
}, 
"previousCommit": { 
"@id": "d3b258d4-c2e3-4385-b044-e32f535ae673" 
}, 
"timestamp": "2021-11-06T17:01:11.031488-04:00" 
}

Element

GET /projects/{projectId}/commits/{commitId}/elements

Gets all the elements at the commit with id {commitId} in the project with id {projectId}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/commits/42122503-70f1-42e9-ac6d-9b1ea2790138/elements" -H "accept: application/json"
Response:

```
[

  {
    "@type": "PartDefinition",
    "@id": "4ace3d89-fd5d-4a03-a303-376eea6fbf29",
    "identifier": "4ace3d89-fd5d-4a03-a303-376eea6fbf29",
    "name": "Vehicle_B",
    ...
  },

  {
    "@type": "PartDefinition",
    "@id": "72f90039-064b-4e91-a21d-7c58813aa4c1",
    "identifier": "72f90039-064b-4e91-a21d-7c58813aa4c1",
    "name": "Vehicle_A",
    ...
  }
]
```

GET /projects/{projectId}/commits/{commitId}/elements/{elementId}

Gets the element with id {elementId} at the commit with id {commitId} in the project with id {projectId}.

Request:
```
curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/commits/42122503-70f1-42e9-ac6d-9b1ea2790138/elements/72f90039-064b-4e91-a21d-7c58813aa4c1" -H "accept: application/json"
```

Response:

```
{
  "@type": "PartDefinition",
  "@id": "72f90039-064b-4e91-a21d-7c58813aa4c1",
  "name": "Vehicle_A",
  ...
}
```

Query

GET /projects/{projectId}/queries

Gets all the queries defined in the project with id {projectId}.

Request:
```
curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/queries" -H "accept: application/json"
```

Response (body):

```
[

  {
    "@id": "1e368830-eee5-404a-be93-490a2b84d28f",
    "@type": "Query",
    "owningProject": {
      "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
    }
  }
]```
"scope": [],
"select": [
  "owner",
  "@type",
  "name",
  "@id"
],
"where": {
  "@type": "PrimitiveConstraint",
  "inverse": false,
  "operator": "=",
  "property": "@type",
  "value": "PartDefinition"
}
},
{
  "@id": "2cfc2123-90c3-48a9-a165-b50d21ab067d",
  "@type": "Query",
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "scope": [],
  "select": [
    "owner",
    "@type",
    "name",
    "@id"
  ],
  "where": {
    "@type": "PrimitiveConstraint",
    "inverse": false,
    "operator": "=",
    "property": "@type",
    "value": "PartDefinition"
  }
},
{
  "@id": "930167d5-0d5e-4f2e-8544-8f66f200e56a",
  "@type": "Query",
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "scope": [],
  "select": [
    "owner",
    "@type",
    "name",
    "@id"
  ],
  "where": {
    "@type": "PrimitiveConstraint",
    "inverse": false,
    "operator": "=",
    "property": "name",
    "value": "Vehicle_A"
  }
},
{
  "@id": "ab894e8b-d6be-4af0-b029-b52058f20b93",
  "@type": "Query",
}
POST /projects/{projectId}/queries

Creates a new query in project with id `{projectId}` using the query data specified in the body of the request.

Request:

curl -X POST "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/queries" -H "accept: application/json" -H "Content-Type: application/json" -d "{ "@type": "Query", "select": [ "name", "@id", "@type", "owner"] , "where": { "@type": "PrimitiveConstraint", "inverse": false, "operator": ",", "property": "name", "value": "Vehicle_A" } }"

Response (body):

{
  "@id": "2cf32123-90c3-49c9-a165-b50d21ab067d",
  "@type": "Query",
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "scope": [],
  "select": [
    "owner",
    "@type",
    "name",
    "@id"
  ],
  "where": {
    "@type": "PrimitiveConstraint",
    "inverse": false,
    "operator": ",",
    "property": "name",
    "value": "PartDefinition"
  }
}
GET /projects/{projectId}/queries/{queryId}

Gets the query with id {queryId} in project with id {projectId}.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/queries/2cfc2123-90c3-48a9-a165-b50d21ab067d" -H "accept: application/json"

Response (body):

```
{
  "@id": "2cfc2123-90c3-48a9-a165-b50d21ab067d",
  "@type": "Query",
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "scope": [],
  "select": [
    "owner",
    "@type",
    "name",
    "@id"
  ],
  "where": {
    "@type": "PrimitiveConstraint",
    "inverse": false,
    "operator": "=",
    "property": "@type",
    "value": "PartDefinition"
  }
}
```

GET /projects/{projectId}/queries/{queryId}/results

Executes the query with id {queryId} in project with id {projectId}. If the commit is not specified ({commitId} query parameter is absent), the query is executed at the head commit in the default branch of the project.

Request:

curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/queries/2cfc2123-90c3-48a9-a165-b50d21ab067d/results?commitId=42122503-70f1-42e9-ac6d-9b1ea2790138" -H "accept: application/json"

Response (body):

```
[
  {
    "@type": "PartDefinition",
    "@id": "4ace3d89-fd5d-4a03-a303-376eea6fbf29",
    "name": "Vehicle_B",
    "owner": null
  },
  {
    "@type": "PartDefinition",
    "@id": "72f90039-064b-4e91-a21d-7c58813aa4c1",
    "name": "Vehicle_A",
    "owner": null
  }
]
```
B.2 Cookbook

The Systems Modeling API Cookbook is a collection of recipes demonstrating patterns and examples for using the Systems Modeling API and Services. Each recipe is a Jupyter notebook (IPython) with a series of API calls using the REST/HTTP PSM of the API.

The Systems Modeling API Cookbook is available as a zip file (*Systems-Modeling-API-Cookbook.zip*) containing the Jupyter (IPython) notebooks with this specification.