Date: January 2023

Systems Modeling Application Programming Interface (API) and Services

Version 1.0
Release 2022-12

Submitted in response to Systems Modeling Language (SysML®) v2 API and Services RFP (ad/2018-06-03) by:

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

Table of Contents

List of Tables

List of Figures

0 Submission Introduction

0.1 Submission Overview

This proposed specification is submitted in response to the Systems Modeling Language (SysML®) v2 API and Services RFP (ad/2018-16-03). It is being submitted along with two other specifications that are being submitted together in response to the Systems Modeling Language (SysML®) v2 Request for Proposals (RFP):

- Kernel Modeling Language (KerML), Version 1.0
- OMG Systems Modeling Language (SysML), Version 2.0

These specifications are all being submitted together because the APIs and services defined in this specification are intended to work with the language metamodels defined in the other two specification. Together, these three specifications respond to the full SysML v2 vision, as captured jointly by the two RFPs.

0.2 Submission Submitters

The following OMG member organizations are jointly submitting this proposed specification:

- 88Solutions Corporation
- GfSE e.V.
- IBM
- INCOSE
- InterCax LLC
- Lockheed Martin Corporation
- Model Driven Solutions, Inc.
- Dassault Système
- PTC
- Simula Research Laboratory AS

The submitters also thankfully acknowledge the support of over 60 other organizations that participated in the SysML v2 Submission Team.

0.3 Submission - Issues to be discussed

The SysML v2 API and Services RFP does not request that any issue be discussed by submitters.

0.4 API and Services Requirements Tables

0.4.1 Mandatory API & Services Requirements Table

Table 1. Mandatory API & Services Requirements Table

Reqt. ID	Reqt. Name	Text
API 1	API and Services Architecture	This group includes general requirements related to the architecture and design of the API.

Reqt. ID	Reqt. Name	Text
API 1.1	API Architecture	Proposals for SysML v2 API and Services shall specify: 1. Platform-independent model (PIM) that specifies the services and the operations provided by the API. Services are collections of operations. Inputs and outputs of each operation shall be specified and be conformant to the SysML v2 meta-model and resulting UML profile [SysML v2]. The platform-independent model shall be defined using an open standard, such as UML2 or IDL, so that it can be used to auto-generate platform-specific bindings. 2. Mappings to bind the platform-independent model (PIM) to each of platform-specific models (PSMs). The mappings shall be defined using an open standard, such as QVT. 3. Platform-specific models (PSMs) as bindings of the PIM to OSLC 3.0 and one or more commonly used technology platforms, such as, but not limited to, REST/HTTP, Java, C#, Javascript, or GraphQL. The platform-specific models shall also include API documentation for each of the services and their operations. Proposals are also encouraged to leverage the latest industry standards and technologies for API specification and cross-language code generation (bindings), such as Apache Thrift, OpenAPI, and Swagger Codegen.

Reqt. ID	Reqt. Name	Text
API 1.2	API Infrastructure Services	Proposals for SysML v2 API and Services shall provide the following infrastructure-level service specifications. • Service discovery to get all the available services • Service access to get the handle of a service **Supporting Information:** Refer to DDS [DDS] and CORBA [CORBA] as examples
API 1.3	Design Constraints	Proposals for SysML v2 API and Services shall allow extensions of the platform-independent model (PIM) and platform-specific bindings (PSMs). **Supporting Information:** Refer to DDS [DDS] and CORBA [CORBA] as examples
API 2	API and Services Conformance	The requirements in this group are related to measuring the conformance level of SysML v2 modeling environments to the platform-specific models (PSMs). SysML v2 modeling environments may implement one or more platform-specific models in the SysML v2 API and Service specification, as described in section 6.2 and API 1.1 requirement above.
API 2.1	API and Services Conformance Criteria	Proposals for SysML v2 API and Services shall provide measurable conformance criteria for all services in the proposal.

Reqt. ID	Reqt. Name	Text
API 2.2	API and Services Conformance Evaluation Method and Test Cases	Proposals for SysML v2 API and Services shall provide an evaluation method and associated test cases to assess conformance for all the services in the proposal.
API 2.3	Functional Thread Conformance	Proposals for SysML v2 API and Services shall provide test cases to assess conformance of SysML v2 modeling environments to functional threads that use a combination of services. Four functional threads are specified below. Proposals can specify test cases for additional functional threads relevant to model-based systems engineering. • FT 1 - Assessing the impact of a requirement change during system development • FT 2 - Investigation and review of system design and relevant analyses after a failure during system operation is reported • FT 3 – Coordination and concurrent development of systems involving multiple disciplines, such as systems, hardware, software, simulation, project management, manufacturing, and operations • FT 4 Tracing system artifacts downstream and upstream during system development and during system operations and maintenance
SV 1	Service Scope, Conditions, and Response Requirements	The requirements in this group are applicable to all the services in the scope of this RFP.

Reqt. ID	Reqt. Name	Text
SV 1.1	Service Scope	Proposals for SysML v2 API and Services shall provide a mechanism to specify the model and its version, or collection of models and their versions, that are in scope for a service request. **Supporting Information:** Example: If the Model Navigation Service is used by a consumer to get all elements of a specific type, then the consumer must have a mechanism to specify a specific model or collection of models in which elements of that type will be searched.
SV 1.2	Pre- and Post-Conditions	Proposals for SysML v2 API and Services shall provide a model-based specification for the pre-conditions and post-conditions for each service. Pre-conditions include the list of conditions that must be true for the service to initiate, and post-conditions include the list of conditions that must be true after the service has responded, whether successfully or unsuccessfully. **Supporting Information:** **E**xample: An example pre-condition for the Model Update Service may be that the model element being updated must exist and not be checked-out (or being modified). An example post-condition for the same service may be that the model must be in a valid state after the service has responded.

Reqt. ID	Reqt. Name	Text
SV 1.3	Standard Response Model	Proposals for SysML v2 API and Services shall provide a Standard Response Model that represents the response structure for all the services in the proposal. A Standard Response Model will make it feasible for service consumers to process service responses in a standard manner. **Supporting Information:** REST/ HTTP APIs provide common response status codes specified by the HTTP protocol [HTTP/1.1].

Reqt. ID	Reqt. Name	Text
SV 2	Model Navigation Service Requirement	Proposals for SysML v2 API and Services shall specify a service to navigate a SysML v2 model, as described in detail below. (a) Define starting point(s) for navigation i. Get model (top-level/root element) ii. Get an element by its unique identifier iii. Get elements by their type in a given scope using direct or recursive strategy, e.g. get elements of type Block directly in a given package, or get elements of type Block recursively in the context of a given package. (b) Get all properties and their values for a given element, including the meta-data for each property such as its name, type, and multiplicity. (c) Get relationships given one or more of the following criteria, including meta-data for each relationship such as its name and type. i. Source element of the relationship ii. Target element of the relationship iii. Type of the relationship **Supporting Information:** The terms element, property, relationship, type, and package used here are based on the latest version of SysML/UML standard.

Reqt. ID	Reqt. Name	Text
SV 3	Model Creation Service Requirement	Proposals for SysML v2 API and Services shall specify a service to create a model element with a unique identifier given the name, type, and owning element for the model element.
SV 4	Model Update Service Requirement	Proposals for SysML v2 API and Services shall specify a service to update model elements, such as updating the name, type, property values, or the owning element.
SV 5	Model Deletion Service Requirement	Proposals for SysML v2 API and Services shall specify a service to delete model elements according to deletion semantics. The deletion semantics will specify other model elements that may need to be deleted or updated to ensure that the model is valid. **Supporting Information:** Example: The deletion semantics for deleting an element may specify deletion of all owned elements and all outgoing/ incoming relationships from/to the given element.

Reqt. ID	Reqt. Name	Text
SV 6	External Relationship Management Services Requirements	The requirements in this group are related to creating, reading, updating, and deleting relationships between elements in a SysML v2 model and elements in non-SysML models and other SysML v2 models. These relations are termed as “external relationships” and are labeled as 2 in Figure 2. Assumption: These set of SysML v2 services assume that there is a service available to access and reference model elements in non-SysML v2 models.
SV 6.1	Get External Relationships	Proposals for SysML v2 API and Services shall specify a service to get all external relationships for a given SysML v2 model element. The type of external relationship may be specified as a filter. **Supporting Information:** Refer to SV 6.5 below for types of external relationships.
SV 6.2	Create External Relationships	Proposals for SysML v2 API and Services shall specify a service to create an external relationship given its type and two model elements, one of which must be a SysML v2 model element. **Supporting Information:** Refer to SV 6.5 below for types of external relationships.

Reqt. ID	Reqt. Name	Text
SV 6.3	Update External Relationships	Proposals for SysML v2 API and Services shall specify a service to update an external relationship, such as updating the name or type of the relationships, or updating the ends of the relationship. **Supporting Information:** Refer to SV 6.5 below for types of external relationships.
SV 6.4	Delete External Relationships	Proposals for SysML v2 API and Services shall specify a service to delete an external relationship.

Reqt. ID	Reqt. Name	Text
SV 6.5	Types and Behaviors of External Relationships	Proposals for SysML v2 API and Services shall specify the types of external relationships and their semantics that includes at least the following: 1. Reference / Trace: Provide navigation between related elements 2. Version tracking: Track versions of the related elements over time 3. Executable model wrapping: Transfer inputs from the source element (at one end) to execute a model (at the other end) and to transfer results from the model execution back to the source element 4. Data transfer: Bi-directionally transfer attribute values from model element at one end to the model element at the other end of the relationship 5. Surrogate: Represent a model element in one domain as a surrogate of a model element in another domain, e.g. represent a physical part in a SysML model as a surrogate of a part in a PLM system 6. Model generation: Generate/ update model (or model element) at one end of the relationship from the model (or model element) at the other end, e.g. generating code from behavior elements in a SysML v2 model, or generating a simulation model from system ports, connectors, and item flows in a SysML v2 model.

0.4.2 Non-Mandatory API & Services Requirements Table

Table 2. Non-Mandatory API & Services Requirements Table

Reqt. ID	Reqt. Name	Text
SVA 1	Model Analysis Services Requirements	The requirements in this group are related to services for creating, querying, updating, deleting, and executing analysis-related model elements in SysML v2 models.
SVA 1.1	Analysis Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting analysis-related model elements by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). **Supporting Information:** Section 6.5.2.8 (requirement group ANL 1) of the SysML v2 language RFP specifies requirements for representing analysis-related concepts in SysML v2 language, such as Analysis, Analysis Model, Analysis Objective, Analysis Scenarios, Analysis Result, and others. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on analysis-related model elements in a SysML v2 model.

Reqt. ID	Reqt. Name	Text
SVA 1.2	Analysis Execution Service	Proposals for SysML v2 API and Services may specify a service to execute analysis models. The service shall consume analysis inputs and produce analysis results conformant to the SysML v2 language. **Supporting Information:** Section 6.5.2.8 (ANL 1.09) of SysML v2 RFP states: Analysis models can be defined natively in SysML (e.g. parametric model or behavior model) or externally (e.g. equation-based math models, finite element analysis models, or computational fluid dynamics models). Example: Consider the following two scenarios for an analysis model to compute the energy collected by an array of solar panels on a spacecraft in a 24-hour period. In the first scenario, the analysis model for energy collection is defined entirely using parametric and activity-related concepts in a SysML v2 model. In the second scenario, the analysis model in a SysML v2 model is only a specification that is linked using an external relationship to a MATLAB function for energy calculation. A SysML v2 modeling environment may support one or both the scenarios and make them available via the Analysis Execution Service API. For the first scenario, the Analysis Execution Service implementation may take the inputs specified in the SysML v2 model, invoke an execution engine provided by the SysML v2 modeling environment to execute the parametric/activity model for energy collection, and return the results as a service response. For the second scenario, the Analysis Execution Service implementation may invoke the

Reqt. ID	Reqt. Name	Text
		MATLAB code for energy calculation with the given inputs in the SysML v2 model and return the output of the MATLAB code execution as a service response.
SVC 1	Advanced Model Construction Services Requirements	The requirements in this group are related to advanced services for creating, updating, and deleting elements in the SysML v2 model beyond the basic model creation services (see requirements SV 3 to SV 6 in section 6.5.2 under Mandatory Requirements.
SVC 1.1	Bulk Model Creation Service	Proposals for SysML v2 API and Services may specify a service to create multiple model elements (in bulk), each with their unique identifier, given the name, type, and owning element for each model element.
SVC 1.2	Bulk Model Update Service	Proposals for SysML v2 API and Services may specify a service to update multiple model elements (in bulk), such as updating the name, type, property values, or the owning element for each model element.

Reqt. ID	Reqt. Name	Text
SVC 1.3	Bulk Model Deletion Service	Proposals for SysML v2 API and Services may specify a service to delete multiple model elements (in bulk) according to deletion semantics. The deletion semantics will specify other model elements that may need to be deleted or updated to ensure that the model is valid. **Supporting Information:** Example: The deletion semantics for deleting an element may specify deletion of all owned elements and all outgoing/ incoming relationships from/to the given element.
SVC 1.4	Bulk External Relationship Management Service	The requirements in this group are related to services for bulk creation, update, and deletion of external relationships.
SVC 1.4.1	Bulk External Relationship Creation Service	Proposals for SysML v2 API and Services may specify a service to create multiple external relationships (in bulk), given the relationship type and two model elements (one of which must be a SysML v2 model element) for each external relationship that needs to be created. **Supporting Information:** Refer to SV 6.5 (section 6.5.2 under Mandatory Requirements) for types of external relationships.

Reqt. ID	Reqt. Name	Text
SVC 1.4.2	Bulk External Relationship Update Service	Proposals for SysML v2 API and Services may specify a service to update multiple external relationships (in bulk), such as updating the name or type or participant model elements for each relationship. **Supporting Information:** Refer to SV 6.5 (section 6.5.2 under Mandatory Requirements) for types of external relationships.
SVC 1.4.3	Bulk External Relationship Deletion Service	Proposals for SysML v2 API and Services may specify a service to delete multiple external relationships (in bulk).
SVG 1	General Services Requirements	The requirements in this group are related to general services, such as timestamp and UUID generation, and API call back.

Reqt. ID	Reqt. Name	Text
SVG 1.1	Timestamp Generation	Proposals for SysML v2 API and Services may specify a service to read and process standard-formatted timestamps on existing model elements and to provide standard-formatted timestamps for new model elements, or new versions of existing model elements. **Supporting Information:** An example of a timestamp, including both date and time (with time zone) is [2009-06-15T13:45:30-04:00]. Refer to ISO 8601 for a standard representation of date and time. The service can create a new timestamp from the current clock time and time zone information, or provide year, month, day, time, and time zone details given a timestamp. Requirement CRC 1.6.2 (section 6.5.2 under Mandatory Requirements) of the SysML v2 language RFP requires that SysML v2 language provide a way to represent timestamp meta-data for model elements.
SVG 1.2	UUID Generation	Proposals for SysML v2 API and Services may specify a service to generate a universally unique identifier (UUID). **Supporting Information:** Requirement CRC 1.2.2 (section 6.5.2 under Mandatory Requirements) of the SysML v2 language RFP requires that SysML v2 language provide a way to represent a universally unique identifier for each model element that cannot be changed.

Reqt. ID	Reqt. Name	Text
SVG 1.3	API Call Back	Proposals for SysML v2 API and Services may specify a service to create or remove callbacks. **Supporting Information:** A callback is an asynchronous, out-of-band request that a service can send to some other services in response to certain events [OpenAPI]. For example, a subscription functionality is a form of a callback where service consumers can subscribe to certain events of a service and receive notification when that event occurs. Most modern APIs provide a way to define callbacks, especially for other services and user-interfaces to respond to events. Callbacks are defined in the OpenAPI specification that submitters are encouraged to leverage for SysML 2 API and Services (see section 6.3).
SVM 1	Model Management Services Requirements	The requirements in this group are related to services for version and configuration management of SysML v2 models, and data control services.
SVM 1.1	Model Versioning Services	The requirements in this group are related to services for version management of SysML v2 models and model elements.

Reqt. ID	Reqt. Name	Text
SVM 1.1.1	Define MCI Definition Default Rules	Proposals for SysML v2 API and Services may specify services to define the rules to determine the type of model elements that will be versioned, which is referred to as a Model Configuration Item (MCI), and types of changes in a MCI that qualify as a version update. **Supporting Information:** A Model Configuration Item is a specific portion of the system model that is maintained in a controlled fashion, i.e. has a unique ID and version history. MCI can be defined in different granularities, from an individual fine-grained Model Element, a set of Model Elements, a set of Elements, to the entire Model. Any MCI can contain another MCI. Examples include Container (i.e. Package), Block, Model Element, and View Definition. Refer to section A.2 (Glossary) for more details. Requirement CRC 1.6.1 (section 6.5.2 under Mandatory Requirements) of the SysML v2 language RFP requires that the SysML v2 language provide a way to represent version meta-data for model elements.

Reqt. ID	Reqt. Name	Text
SVM 1.1.2	Create and Delete Versions of a MCI	Proposals for SysML v2 API and Services may specify services that can create new versions or delete a specific version of a Model Configuration Item. **Supporting Information:** Services for model construction will use this service to create new versions or delete existing versions of model elements. For example, Model Creation Service (SV 3, section 6.5.2) will require this service to create the first version of a model element if it is a MCI. Model Update Service (SV 4, section 6.5.2) will require this service to create a new version of an existing model element if it is a MCI.
SVM 1.1.3	Get Versions of a MCI	Proposals for SysML v2 API and Services may specify services to get the version history of a MCI. The version history provides a chronological list of all the versions of the given MCI.
SVM 1.1.4	Compare Versions of a MCI	Proposals for SysML v2 API and Services may specify services to compare two or more versions of a MCI and provide a list of differences in a standard way.

Reqt. ID	Reqt. Name	Text
SVM 1.2	Model Configuration Services	Proposals for SysML v2 API and Services may specify services to create baseline configurations with MCIs, create branch configurations with MCIs from a given baseline configuration, and merge branch configurations with each other or with the originating baseline configuration. **Supporting Information:** The intent of this service (or set of services) is to provide a standard way to define stable releases of a SysML v2 model as baselines, and allow multiple parallel branches of concurrent development on a SysML v2 model.
SVM 1.3	Model Data Control Services	Proposals for SysML v2 API and Services may specify services to create, read, update, and delete data protection control markings on model elements. **Supporting Information:** Requirement CRC 1.6.3 (section 6.5.2 under Mandatory Requirements) of the SysML v2 language RFP requires that SysML v2 language provide a way to represent data protection control markings on model elements, such as ITAR, security, and proprietary classifications.

Reqt. ID	Reqt. Name	Text
SVQ 1	Model Query Services Requirements	The requirements in this group are related to services for creating, executing, and managing queries for SysML v2 models, beyond basic model navigation services covered in SV 2 (section 6.5.2 under Mandatory Requirements). **Supporting Information:** A query is a precise request for information retrieval from a SysML v2 model (see section A.2 - Glossary).

Reqt. ID	Reqt. Name	Text
SVQ 1.1	Standard Query Model	Proposals for SysML v2 API and Services may provide a Standard Query Model to specify the scope, input criteria, and the outputs requested in a query. The elements used to specify the input criteria and the outputs shall conform to the SysML v2 meta-model and the resulting UML profile. The Standard Query Model shall serve as a language- and platform-independent interface definition for a query to a SysML v2 model, and can be used to generate language- or platform-specific queries for SysML v2 modeling environments. **Supporting Information:** **E**xample: Consider an example query to get all electrical interfaces in system S, where electrical interfaces are identified as ports typed by standard value types AC or DC. A query can be created conforming to the Standard Query Model—scope of the query is System S (immediate level or recursive), the input criteria is port type is AC or port type is DC, and the requested output element type is Port. Now consider two different SysML v2 modeling environments, one that provides a REST/HTTP binding (PSM implementation) and one that provides a SQL binding. The Standard Query Model can be mapped to both the bindings, e.g. GET calls to one or more REST API endpoints in one SysML v2 modeling environment and SQL-based query in the other SysML v2 modeling environment.

Reqt. ID	Reqt. Name	Text
SVQ 1.2	Query Execution Service	Proposals for SysML v2 API and Services may specify a service to execute queries specified using the Standard Query Model. **Supporting Information:** Example: Consider two different SysML v2 modeling environments that implement the query execution service based on their specific platform bindings (REST/ HTTP versus SQL/JDBC). In one implementation, the query execution service is implemented as a REST endpoint that accepts a query specified using the Standard Query Model in JSON format and returns result conforming to the Standard Response Model (SV 1.1, section 6.5.2) in JSON format. In the other implementation, the query execution service is implemented using a Java-based JDBC call that accepts a query specified using the Standard Query Model, converts it to a SQL-based query expression, runs the SQL query, and returns the query result conforming to the Standard Response Model as a Java object.
SVT 1	Model Transformation Services Requirements	The requirements in this group are related to services for transforming SysML models.
SVT 1.1	Model Transformation Service	Proposals for SysML v2 may specify services to create, read, update, delete, and execute model transformations specified as SysML models.

Reqt. ID	Reqt. Name	Text
SVT 1.2	SysML Version to Version Transformation	Proposals for SysML v2 may provide services to transform a model in a previous version of SysML to a model in the next version of SysML, beginning with the transformation from the current version of SysML v1 to SysML v2. Proposals may provide services to transform models in earlier versions of SysML v1 (e.g. 1.3 and 1.4 if 1.5 is the current version) to SysML v2 models. Proposals must state the specific SysML v1 versions supported beyond the current version. **Supporting Information:** This includes the ability to transform the abstract syntax, concrete syntax and semantics. Some of the SysML v2 execution semantics are specified in other specifications including fUML, PSCS, and PSSM.
SVT 1.3	Model to Textual Syntax Generation Service	Proposals for SysML v2 API and Services may provide a service to generate the textual representation of a SysML v2 model (or model elements) conforming to the concrete textual syntax of the SysML v2 language. The scope of model elements shall be specified as an input to the service. **Supporting Information:** The concrete textual syntax for SysML v2 language is presented in requirement LNG 1.4.4 of the SysML v2 RFP under Non-Mandatory Features. The concrete textual syntax provides a textual human-readable representation of a SysML v2 model.

Reqt. ID	Reqt. Name	Text
SVT 1.4	Textual Syntax to Model Generation Service	Proposals for SysML v2 API and Services may provide a service to generate SysML v2 models (or model elements) from a textual representation conforming to the concrete textual syntax of the SysML v2 language. **Supporting Information:** The concrete textual syntax for SysML v2 language is presented in requirement LNG 1.4.4 of the SysML v2 RFP under Non-Mandatory Features. The concrete textual syntax provides a textual human-readable representation of a SysML v2 model.
SVV 1	Model View and Viewpoint Management Services Requirements	The requirements in this group are related to services for creating, querying, updating, and deleting viewpoints and views in SysML v2 models.
SVV 1.1	Viewpoint Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting viewpoints by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). **Supporting Information:** Section 6.5.2.1 (requirement group CRC 1.5) of the SysML v2 language RFP specifies requirements for representing view and viewpoint-related concepts in SysML v2 language. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on viewpoint-related model elements in a SysML v2 model.

Reqt. ID	Reqt. Name	Text
SVV 1.2	View Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting views by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). Services to create, query, update, and delete views shall provide a mechanism to specify the viewpoint—to which the subject views conform—as an input. **Supporting Information:** Section 6.5.2.1 (requirement group CRC 1.5) of the SysML v2 language RFP specifies requirements for representing view and viewpoint-related concepts in SysML v2 language. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on view-related model elements in a SysML v2 model.

0.4.3 Mandatory API and Services Requirements - Satisfied-by Table

Table 3. Mandatory API and Services Requirements - Satisfied-by Table

ID	Name	Satisfied?	Satisfied-by	Comment
API 1	API and Services Architecture	Yes		
API 1.1	API Architecture	Yes	1 Platform Independent Model (PIM) 2 Platform Specific Models (PSMs)	
API 1.2	API Infrastructure Services	Partial	1 Platform Independent Model (PIM)	
API 1.3	Design Constraints	Yes	1 Platform Independent Model (PIM) 2 Platform Specific Models (PSMs)	

ID	Name	Satisfied?	Satisfied-by	Comment
API 2	API and Services Conformance	Yes		
API 2.1	API and Services Conformance Criteria	Yes	Annex A - Conformance Test Cases	
API 2.2	API and Services Conformance Evaluation Method and Test Cases	Yes	Annex A - Conformance Test Cases	
API 2.3	Functional Thread Conformance	Yes	Annex A - Conformance Test Cases	
SV 1	Service Scope, Conditions, and Response Requirements	Yes		
SV 1.1	Service Scope	Yes	1 Platform Independent Model (PIM)	
SV 1.2	Pre- and Post-Conditions	Yes	1 Platform Independent Model (PIM) Annex A - Conformance Test Cases	
SV 1.3	Standard Response Model	Yes	1 Platform Independent Model (PIM)	
SV 2	Model Navigation Service Requirement	Yes	ElementNavigationService QueryService	
SV 3	Model Creation Service Requirement	Yes	ProjectDataVersioningService	
SV 4	Model Update Service Requirement	Yes	ProjectDataVersioningService	
SV 5	Model Deletion Service Requirement	Yes	ProjectDataVersioningService	
SV 6	External Relationship Management Services Requirements	Yes	ProjectDataVersioningService ExternalRelationship ExternalData	ExternalRelationship and ExternalData are defined in the SysML v2 API as realizations of the Data interface. The geneic services for data creation, update, and deletion apply.

ID	Name	Satisfied?	Satisfied-by	Comment
SV 6.1	Get External Relationships	Yes	QueryService ElementNavigationService	
SV 6.2	Create External Relationships	Yes	ProjectDataVersioningService	
SV 6.3	Update External Relationships	Yes	ProjectDataVersioningService	
SV 6.4	Delete External Relationships	Yes	ProjectDataVersioningService	
SV 6.5	Types and Behaviors of External Relationships	Yes	ExternalRelationship	

0.4.4 Non-Mandatory API and Services Requirements - Satisfied-by Table

Table 4. Non-Mandatory API and Services Requirements - Satisfied-by Table

ID	Name	Satisfied?	Satisfied-by	Comment
SVA 1	Model Analysis Services Requirements	Yes		
SVA 1.1	Analysis Creation, Query, Update, and Deletion Services	Yes	ProjectDataVersioningService ElementNavigationService QueryService	Analysis-related model elements extend Element and Relationship in SysML v2 meta-model. The same creation, query, update, and deletion services apply.
SVA 1.2	Analysis Execution Service	No		
SVC 1	Advanced Model Construction Services Requirements	Yes		SysML v2 API consumers can request creation, update, and deletion of multiple elements in a single commit change set.
SVC 1.1	Bulk Model Creation Service	Yes	ProjectDataVersioningService	
SVC 1.2	Bulk Model Update Service	Yes	ProjectDataVersioningService	
SVC 1.3	Bulk Model Deletion Service	Yes	ProjectDataVersioningService	

ID	Name	Satisfied?	Satisfied-by	Comment
SVC 1.4	Bulk External Relationship Management Service	Yes	ProjectDataVersioningService	
SVC 1.4.1	Bulk External Relationship Creation Service	Yes	ProjectDataVersioningService	
SVC 1.4.2	Bulk External Relationship Update Service	Yes	ProjectDataVersioningService	
SVC 1.4.3	Bulk External Relationship Deletion Service	Yes	ProjectDataVersioningService	
SVG 1	General Services Requirements	Yes		
SVG 1.1	Timestamp Generation	Yes	1 Platform Independent Model (PIM) 2 Platform Specific Models (PSMs)	
SVG 1.2	UUID Generation	Yes	1 Platform Independent Model (PIM) 2 Platform Specific Models (PSMs)	
SVG 1.3	API Call Back	No		
SVM 1	Model Management Services Requirements	Yes	ProjectDataVersioningService	
SVM 1.1	Model Versioning Services	Yes	ProjectDataVersioningService	
SVM 1.1.1	Define MCI Definition Default Rules	Yes	ProjectDataVersioningService DataIdentity DataVersion Data	
SVM 1.1.2	Create and Delete Versions of a MCI	Yes	ProjectDataVersioningService	
SVM 1.1.3	Get Versions of a MCI	Yes	ProjectDataVersioningService	
SVM 1.1.4	Compare Versions of a MCI	No		
SVM 1.2	Model Configuration Services	Yes	ProjectDataVersioningService	

ID	Name	Satisfied?	Satisfied-by	Comment
SVM 1.3	Model Data Control Services	Yes	ProjectDataVersioningService	Data protection control markings can be represented as: (1) Direct realization of the Data interface provided by the SysML v2 API, or (2) Extensions of Element concept in SysML v2 meta-model. The general services for Project and Data versioning will apply.
SVQ 1	Model Query Services Requirements	Yes		
SVQ 1.1	Standard Query Model	Yes	QueryService Query	
SVQ 1.2	Query Execution Service	Yes	QueryService Query	
SVT 1	Model Transformation Services Requirements	No		
SVT 1.1	Model Transformation Service	No		
SVT 1.2	SysML Version to Version Transformation	No		
SVT 1.3	Model to Textual Syntax Generation Service	No		
SVT 1.4	Textual Syntax to Model Generation Service	No		

ID	Name	Satisfied?	Satisfied-by	Comment
SVV 1	Model View and Viewpoint Management Services Requirements	No		View and Viewpoint-related model elements extend Element and Relationship in SysML v2 meta-model. The same creation, query, update, and deletion services apply.
SVV 1.1	Viewpoint Creation, Query, Update, and Deletion Services	Yes	ProjectDataVersioningService QueryService ElementNavigationService	
SVV 1.2	View Creation, Query, Update, and Deletion Services	Yes	ProjectDataVersioningService QueryService ElementNavigationService	

0.4.5 Changed API and Services Requirements Table

Table 5. Changed API and Services Requirements Table

ID	Name	Requirement Text	Change Status	Change Description
API 2	API and Services Conformance	The requirements in this group are related to measuring the conformance level of SysML v2 modeling environments to the platform-specific models (PSMs). SysML v2 modeling environments may implement one or more platform-specific models in the SysML v2 API and Service specification, as described in section 6.2 and API 1.1 requirement above.	Modified	

ID	Name	Requirement Text	Change Status	Change Description
API 2.2	API and Services Conformance Evaluation Method and Test Cases	Proposals for SysML v2 API and Services shall provide an evaluation method and associated test cases to assess conformance for all the services in the proposal.	Modified	

ID	Name	Requirement Text	Change Status	Change Description
API 2.3	Functional Thread Conformance	Proposals for SysML v2 API and Services shall provide test cases to assess conformance of SysML v2 modeling environments to functional threads that use a combination of services. Four functional threads are specified below. Proposals can specify test cases for additional functional threads relevant to model-based systems engineering. • FT 1 - Assessing the impact of a requirement change during system development • FT 2 - Investigation and review of system design and relevant analyses after a failure during system operation is reported • FT 3 – Coordination and concurrent development of systems involving multiple disciplines, such as systems, hardware, software, simulation, project management, manufacturing, and operations • FT 4 Tracing system artifacts downstream and upstream during	Modified	

ID	Name	Requirement Text	Change Status	Change Description
		system development and during system operations and maintenance		

ID	Name	Requirement Text	Change Status	Change Description
SV 6.5	Types and Behaviors of External Relationships	Proposals for SysML v2 API and Services shall specify the types of external relationships and their semantics that includes at least the following: 1. Reference / Trace: Provide navigation between related elements 2. Version tracking: Track versions of the related elements over time 3. Executable model wrapping: Transfer inputs from the source element (at one end) to execute a model (at the other end) and to transfer results from the model execution back to the source element 4. Data transfer: Bi-directionally transfer attribute values from model element at one end to the model element at the other end of the relationship 5. Surrogate: Represent a model element in one domain as a surrogate of a model element in another domain, e.g. represent a physical part in a SysML model as a surrogate	Modified	

ID	Name	Requirement Text	Change Status	Change Description
		of a part in a PLM system 6. Model generation: Generate/update model (or model element) at one end of the relationship from the model (or model element) at the other end, e.g. generating code from behavior elements in a SysML v2 model, or generating a simulation model from system ports, connectors, and item flows in a SysML v2 model.		
SVA 1	Model Analysis Services Requirements	The requirements in this group are related to services for creating, querying, updating, deleting, and executing analysis-related model elements in SysML v2 models.	Modified	30 June 2018 - The original RFP requirement number was SVA

ID	Name	Requirement Text	Change Status	Change Description
SVA 1.1	Analysis Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting analysis-related model elements by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). **Supporting Information:** Section 6.5.2.8 (requirement group ANL 1) of the SysML v2 language RFP specifies requirements for representing analysis-related concepts in SysML v2 language, such as Analysis, Analysis Model, Analysis Objective, Analysis Scenarios, Analysis Result, and others. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on analysis-related model elements in a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVA1

ID	Name	Requirement Text	Change Status	Change Description
SVA 1.2	Analysis Execution Service	Proposals for SysML v2 API and Services may specify a service to execute analysis models. The service shall consume analysis inputs and produce analysis results conformant to the SysML v2 language. **Supporting Information:** Section 6.5.2.8 (ANL 1.09) of SysML v2 RFP states: Analysis models can be defined natively in SysML (e.g. parametric model or behavior model) or externally (e.g. equation-based math models, finite element analysis models, or computational fluid dynamics models). Example: Consider the following two scenarios for an analysis model to compute the energy collected by an array of solar panels on a spacecraft in a 24-hour period. In the first scenario, the analysis model for energy collection is defined entirely using parametric and activity-related concepts in a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVA2

ID	Name	Requirement Text	Change Status	Change Description
		In the second scenario, the analysis model in a SysML v2 model is only a specification that is linked using an external relationship to a MATLAB function for energy calculation. A SysML v2 modeling environment may support one or both the scenarios and make them available via the Analysis Execution Service API. For the first scenario, the Analysis Execution Service implementation may take the inputs specified in the SysML v2 model, invoke an execution engine provided by the SysML v2 modeling environment to execute the parametric/activity model for energy collection, and return the results as a service response. For the second scenario, the Analysis Execution Service implementation may invoke the MATLAB code for energy calculation with the given inputs in the		

ID	Name	Requirement Text	Change Status	Change Description
		SysML v2 model and return the output of the MATLAB code execution as a service response.		
SVC 1	Advanced Model Construction Services Requirements	The requirements in this group are related to advanced services for creating, updating, and deleting elements in the SysML v2 model beyond the basic model creation services (see requirements SV 3 to SV 6 in section 6.5.2 under Mandatory Requirements.	Modified	30 June 2018 - The original RFP requirement number was SVC
SVC 1.1	Bulk Model Creation Service	Proposals for SysML v2 API and Services may specify a service to create multiple model elements (in bulk), each with their unique identifier, given the name, type, and owning element for each model element.	Modified	30 June 2018 - The original RFP requirement number was SVC1
SVC 1.2	Bulk Model Update Service	Proposals for SysML v2 API and Services may specify a service to update multiple model elements (in bulk), such as updating the name, type, property values, or the owning element for each model element.	Modified	30 June 2018 - The original RFP requirement number was SVC2

ID	Name	Requirement Text	Change Status	Change Description
SVC 1.3	Bulk Model Deletion Service	Proposals for SysML v2 API and Services may specify a service to delete multiple model elements (in bulk) according to deletion semantics. The deletion semantics will specify other model elements that may need to be deleted or updated to ensure that the model is valid. **Supporting Information:** Example: The deletion semantics for deleting an element may specify deletion of all owned elements and all outgoing/ incoming relationships from/ to the given element.	Modified	30 June 2018 - The original RFP requirement number was SVC3
SVC 1.4	Bulk External Relationship Management Service	The requirements in this group are related to services for bulk creation, update, and deletion of external relationships.	Modified	30 June 2018 - The original RFP requirement number was SVC4

ID	Name	Requirement Text	Change Status	Change Description
SVC 1.4.1	Bulk External Relationship Creation Service	Proposals for SysML v2 API and Services may specify a service to create multiple external relationships (in bulk), given the relationship type and two model elements (one of which must be a SysML v2 model element) for each external relationship that needs to be created. **Supporting Information:** Refer to SV 6.5 (section 6.5.2 under Mandatory Requirements) for types of external relationships.	Modified	30 June 2018 - The original RFP requirement number was SVC4.1
SVC 1.4.2	Bulk External Relationship Update Service	Proposals for SysML v2 API and Services may specify a service to update multiple external relationships (in bulk), such as updating the name or type or participant model elements for each relationship. **Supporting Information:** Refer to SV 6.5 (section 6.5.2 under Mandatory Requirements) for types of external relationships.	Modified	30 June 2018 - The original RFP requirement number was SVC4.2

ID	Name	Requirement Text	Change Status	Change Description
SVC 1.4.3	Bulk External Relationship Deletion Service	Proposals for SysML v2 API and Services may specify a service to delete multiple external relationships (in bulk).	Modified	30 June 2018 - The original RFP requirement number was SVC4.3
SVG 1	General Services Requirements	The requirements in this group are related to general services, such as timestamp and UUID generation, and API call back.	Modified	30 June 2018 - The original RFP requirement number was SVG
SVM 1	Model Management Services Requirements	The requirements in this group are related to services for version and configuration management of SysML v2 models, and data control services.	Modified	30 June 2018 - The original RFP requirement number was SVM
SVM 1.1	Model Versioning Services	The requirements in this group are related to services for version management of SysML v2 models and model elements.	Modified	30 June 2018 - The original RFP requirement number was SVM1

ID	Name	Requirement Text	Change Status	Change Description
SVM 1.1.1	Define MCI Definition Default Rules	Proposals for SysML v2 API and Services may specify services to define the rules to determine the type of model elements that will be versioned, which is referred to as a Model Configuration Item (MCI), and types of changes in a MCI that qualify as a version update. **Supporting Information:** A Model Configuration Item is a specific portion of the system model that is maintained in a controlled fashion, i.e. has a unique ID and version history. MCI can be defined in different granularities, from an individual fine-grained Model Element, a set of Model Elements, a set of Elements, to the entire Model. Any MCI can contain another MCI. Examples include Container (i.e. Package), Block, Model Element, and View Definition. Refer to section A.2 (Glossary) for more details. Requirement CRC 1.6.1 (section 6.5.2 under Mandatory	Modified	30 June 2018 - The original RFP requirement number was SVM1.1

ID	Name	Requirement Text	Change Status	Change Description
		Requirements) of the SysML v2 language RFP requires that the SysML v2 language provide a way to represent version meta-data for model elements.		
SVM 1.1.2	Create and Delete Versions of a MCI	Proposals for SysML v2 API and Services may specify services that can create new versions or delete a specific version of a Model Configuration Item. **Supporting Information:** Services for model construction will use this service to create new versions or delete existing versions of model elements. For example, Model Creation Service (SV 3, section 6.5.2) will require this service to create the first version of a model element if it is a MCI. Model Update Service (SV 4, section 6.5.2) will require this service to create a new version of an existing model element if it is a MCI.	Modified	30 June 2018 - The original RFP requirement number was SVM1.2

ID	Name	Requirement Text	Change Status	Change Description
SVM 1.1.3	Get Versions of a MCI	Proposals for SysML v2 API and Services may specify services to get the version history of a MCI. The version history provides a chronological list of all the versions of the given MCI.	Modified	30 June 2018 - The original RFP requirement number was SVM1.3
SVM 1.1.4	Compare Versions of a MCI	Proposals for SysML v2 API and Services may specify services to compare two or more versions of a MCI and provide a list of differences in a standard way.	Modified	30 June 2018 - The original RFP requirement number was SVM1.4

ID	Name	Requirement Text	Change Status	Change Description
SVM 1.2	Model Configuration Services	Proposals for SysML v2 API and Services may specify services to create baseline configurations with MCIs, create branch configurations with MCIs from a given baseline configuration, and merge branch configurations with each other or with the originating baseline configuration. **Supporting Information:** The intent of this service (or set of services) is to provide a standard way to define stable releases of a SysML v2 model as baselines, and allow multiple parallel branches of concurrent development on a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVM2

ID	Name	Requirement Text	Change Status	Change Description
SVM 1.3	Model Data Control Services	Proposals for SysML v2 API and Services may specify services to create, read, update, and delete data protection control markings on model elements. **Supporting Information:** Requirement CRC 1.6.3 (section 6.5.2 under Mandatory Requirements) of the SysML v2 language RFP requires that SysML v2 language provide a way to represent data protection control markings on model elements, such as ITAR, security, and proprietary classifications.	Modified	30 June 2018 - The original RFP requirement number was SVM3

ID	Name	Requirement Text	Change Status	Change Description
SVQ 1	Model Query Services Requirements	The requirements in this group are related to services for creating, executing, and managing queries for SysML v2 models, beyond basic model navigation services covered in SV 2 (section 6.5.2 under Mandatory Requirements). **Supporting Information:** A query is a precise request for information retrieval from a SysML v2 model (see section A.2 - Glossary).	Modified	30 June 2018 - The original RFP requirement number was SVQ

ID	Name	Requirement Text	Change Status	Change Description
SVQ 1.1	Standard Query Model	Proposals for SysML v2 API and Services may provide a Standard Query Model to specify the scope, input criteria, and the outputs requested in a query. The elements used to specify the input criteria and the outputs shall conform to the SysML v2 meta-model and the resulting UML profile. The Standard Query Model shall serve as a language- and platform-independent interface definition for a query to a SysML v2 model, and can be used to generate language- or platform-specific queries for SysML v2 modeling environments. **Supporting Information:** **E**xample: Consider an example query to get all electrical interfaces in system S, where electrical interfaces are identified as ports typed by standard value types AC or DC. A query can be created conforming to the Standard Query Model—scope of	Modified	30 June 2018 - The original RFP requirement number was SVQ1

ID	Name	Requirement Text	Change Status	Change Description
		the query is System S (immediate level or recursive), the input criteria is port type is AC or port type is DC, and the requested output element type is Port. Now consider two different SysML v2 modeling environments, one that provides a REST/HTTP binding (PSM implementation) and one that provides a SQL binding. The Standard Query Model can be mapped to both the bindings, e.g. GET calls to one or more REST API endpoints in one SysML v2 modeling environment and SQL-based query in the other SysML v2 modeling environment.		

ID	Name	Requirement Text	Change Status	Change Description
SVQ 1.2	Query Execution Service	Proposals for SysML v2 API and Services may specify a service to execute queries specified using the Standard Query Model. **Supporting Information:** Example: Consider two different SysML v2 modeling environments that implement the query execution service based on their specific platform bindings (REST/HTTP versus SQL/ JDBC). In one implementation, the query execution service is implemented as a REST endpoint that accepts a query specified using the Standard Query Model in JSON format and returns result conforming to the Standard Response Model (SV 1.1, section 6.5.2) in JSON format. In the other implementation, the query execution service is implemented using a Java-based JDBC call that accepts a query specified using the Standard Query Model, converts it to a	Modified	30 June 2018 - The original RFP requirement number was SVQ2

ID	Name	Requirement Text	Change Status	Change Description
		SQL-based query expression, runs the SQL query, and returns the query result conforming to the Standard Response Model as a Java object.		
SVT 1	Model Transformation Services Requirements	The requirements in this group are related to services for transforming SysML models.	Modified	30 June 2018 - The original RFP requirement number was SVT
SVT 1.1	Model Transformation Service	Proposals for SysML v2 may specify services to create, read, update, delete, and execute model transformations specified as SysML models.	Modified	30 June 2018 - The original RFP requirement number was SVT1

ID	Name	Requirement Text	Change Status	Change Description
SVT 1.2	SysML Version to Version Transformation	Proposals for SysML v2 may provide services to transform a model in a previous version of SysML to a model in the next version of SysML, beginning with the transformation from the current version of SysML v1 to SysML v2. Proposals may provide services to transform models in earlier versions of SysML v1 (e.g. 1.3 and 1.4 if 1.5 is the current version) to SysML v2 models. Proposals must state the specific SysML v1 versions supported beyond the current version. **Supporting Information:** This includes the ability to transform the abstract syntax, concrete syntax and semantics. Some of the SysML v2 execution semantics are specified in other specifications including fUML, PSCS, and PSSM.	Modified	30 June 2018 - The original RFP requirement number was SVT2

ID	Name	Requirement Text	Change Status	Change Description
SVT 1.3	Model to Textual Syntax Generation Service	Proposals for SysML v2 API and Services may provide a service to generate the textual representation of a SysML v2 model (or model elements) conforming to the concrete textual syntax of the SysML v2 language. The scope of model elements shall be specified as an input to the service. **Supporting Information:** The concrete textual syntax for SysML v2 language is presented in requirement LNG 1.4.4 of the SysML v2 RFP under Non-Mandatory Features. The concrete textual syntax provides a textual human-readable representation of a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVT3

ID	Name	Requirement Text	Change Status	Change Description
SVT 1.4	Textual Syntax to Model Generation Service	Proposals for SysML v2 API and Services may provide a service to generate SysML v2 models (or model elements) from a textual representation conforming to the concrete textual syntax of the SysML v2 language. **Supporting Information:** The concrete textual syntax for SysML v2 language is presented in requirement LNG 1.4.4 of the SysML v2 RFP under Non-Mandatory Features. The concrete textual syntax provides a textual human-readable representation of a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVT4
SVV 1	Model View and Viewpoint Management Services Requirements	The requirements in this group are related to services for creating, querying, updating, and deleting viewpoints and views in SysML v2 models.	Modified	30 June 2018 - The original RFP requirement number was SVV

ID	Name	Requirement Text	Change Status	Change Description
SVV 1.1	Viewpoint Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting viewpoints by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). **Supporting Information:** Section 6.5.2.1 (requirement group CRC 1.5) of the SysML v2 language RFP specifies requirements for representing view and viewpoint-related concepts in SysML v2 language. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on viewpoint-related model elements in a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVV1

ID	Name	Requirement Text	Change Status	Change Description
SVV 1.2	View Creation, Query, Update, and Deletion Services	Proposals for SysML v2 API and Services may specify services for creating, querying, updating, and deleting views by reusing the model construction services (sections 6.5.2 and 6.6.2) and model navigation and query services (sections 6.5.2 and 6.6.1). Services to create, query, update, and delete views shall provide a mechanism to specify the viewpoint—to which the subject views conform—as an input. **Supporting Information:** Section 6.5.2.1 (requirement group CRC 1.5) of the SysML v2 language RFP specifies requirements for representing view and viewpoint-related concepts in SysML v2 language. The generic services for model navigation, model query, and model construction can be used for providing CRUD operations on view-related model elements in a SysML v2 model.	Modified	30 June 2018 - The original RFP requirement number was SVV2

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

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

1. **PSM-level Conformance** - A Service Provider tool demonstrating PSM-level Conformance implements one or more of the Systems Modeling API and Services PSMs defined in this specification. For example, a Provider tool can implement the REST/HTTP PSM, the OSLC PSM, or both. PSM-level conformance of Service Provider tools ensures interoperability of Service Consumer tools using the PSM across the different Service Providers. See Clause 8.
2. **PIM-level Conformance** - A Service Provider tool 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 tool 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 tool must implement all the operations in the Project Service, 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 tool must implement all the operations in the Element Navigation Service, and demonstrate that the implementation successfully passes all the ElementNavigationService Conformance Test Cases (see A.2) and Cross-Cutting Conformance Test Cases (see A.7).
3. **ProjectDataVersioningService Conformance** - A Service Provider tool must implement all the operations in the Project and Data Versioning Service, 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 tool 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 tool 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 tool 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 tool must implement all the operations in the Query Service, 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 tool must implement all the operations in the Query Service, 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

[SE Handbook] *INCOSE Systems Engineering Handbook*
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/

[SQL] *ISO/IEC 9075:2016, Information technology — Database languages — SQL*
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*
https://tools.ietf.org/html/rfc4122

[XMI] *XML Metadata Interchange (XMI®)*, Version 2.5.1
https://www.omg.org/spec/XMI/2.5.1

4 Terms and Definitions

There are no terms and definitions specific to this specification.

5 Symbols

There are no symbols defined in this specification.

6 Introduction

6.1 API and Services Architecture

The Systems Modeling API and Services includes the following (see Fig. 1):

(1) Platform-Independent Model (PIM) provides a service specification that is consistent with KerML and SysML. The PIM serves as the logical API model and provides a specification of services independent of the platform or technology.

(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.

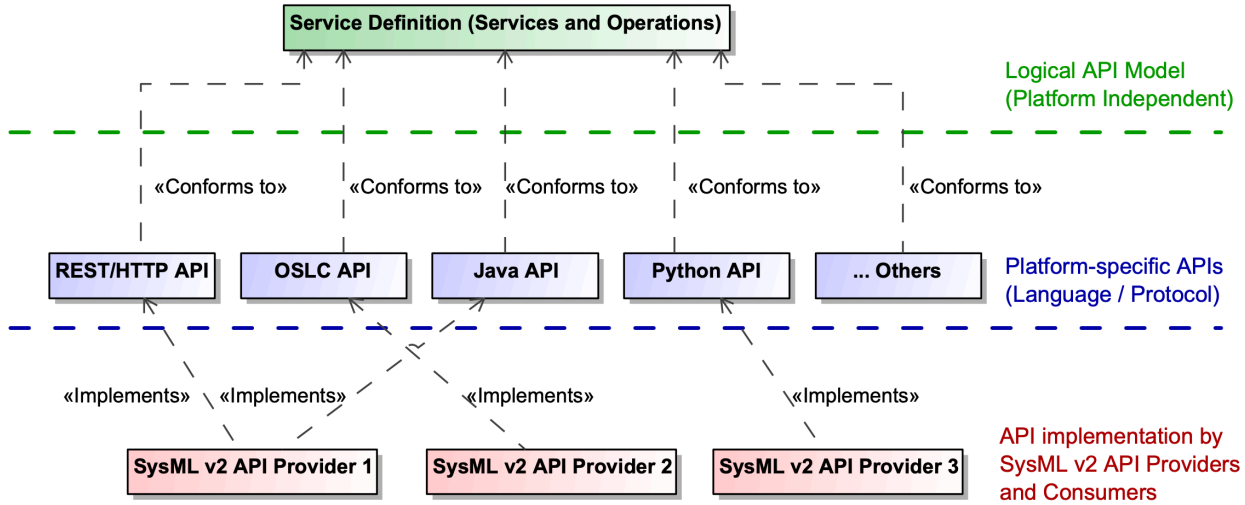

Figure 1. API and Services Architecture

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

Applications written using a platform-specific binding (PSM) should be interoperable across multiple SysML modeling environments that implement the binding, without requiring any modification to the application.

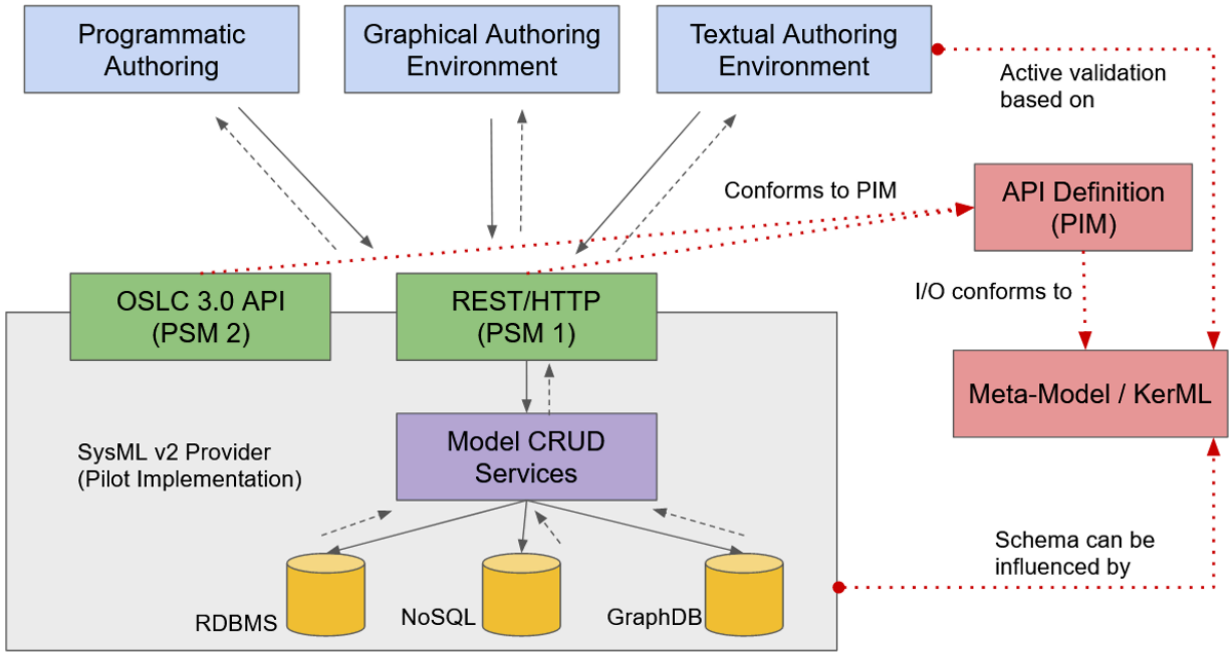

Figure 2. Use of PIM and PSMs by Providers and Consumers

Fig. 2 illustrates the role of PIM and PSMs in the context of Systems Modeling API and Services provider and consumer tools. Version 1 of the Systems Modeling API and Services includes two PSMs, specifically the REST/HTTP PSM and OSLC 3.0 PSM. The inputs and outputs of model create, read, update, and delete (CRUD) operations defined in PIM services conform to the concepts defined in KerML, which are inherited by the SysML v2 meta-model. A System Modeling API and Services provider tool implements either or both the PSMs using its native technology stack, such as databases and web-service frameworks. Consumer tools, such as those used for programmatic, graphical, or textual authoring, navigation, and querying models 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, such as ElementNavigationService.
2. Names of operations representing the API calls available for each service start with a lowercase letter, such as getElementById

Input / 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 Element
2. Names of attributes representing the details of the data that is the input or output of services start with a lowercase letter, 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. In the description, the properties of the input and output data (types) are *italicized.*

The camel case notation is used consistently for the names of services, service operations, operation arguments and return types, and properties of types.

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 an annex.

- Annex A defines the suite of conformance tests that may be used to demonstrate the conformance of systems modeling environments to this specification - see Clause 2.

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

However, work on the specification was also supported by over 120 people in over 60 other 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), 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

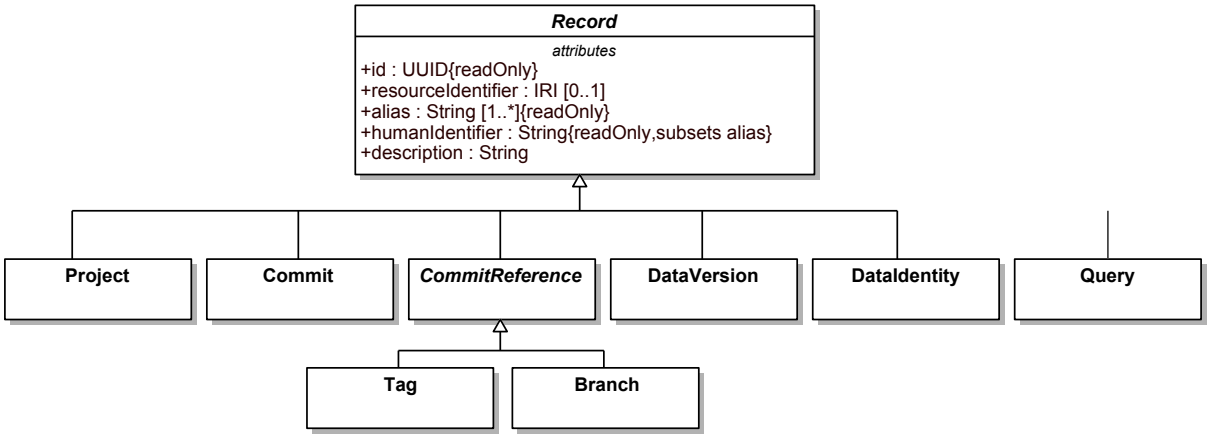

Figure 3. Types of Records

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.

7.1.2 Project Data Versioning

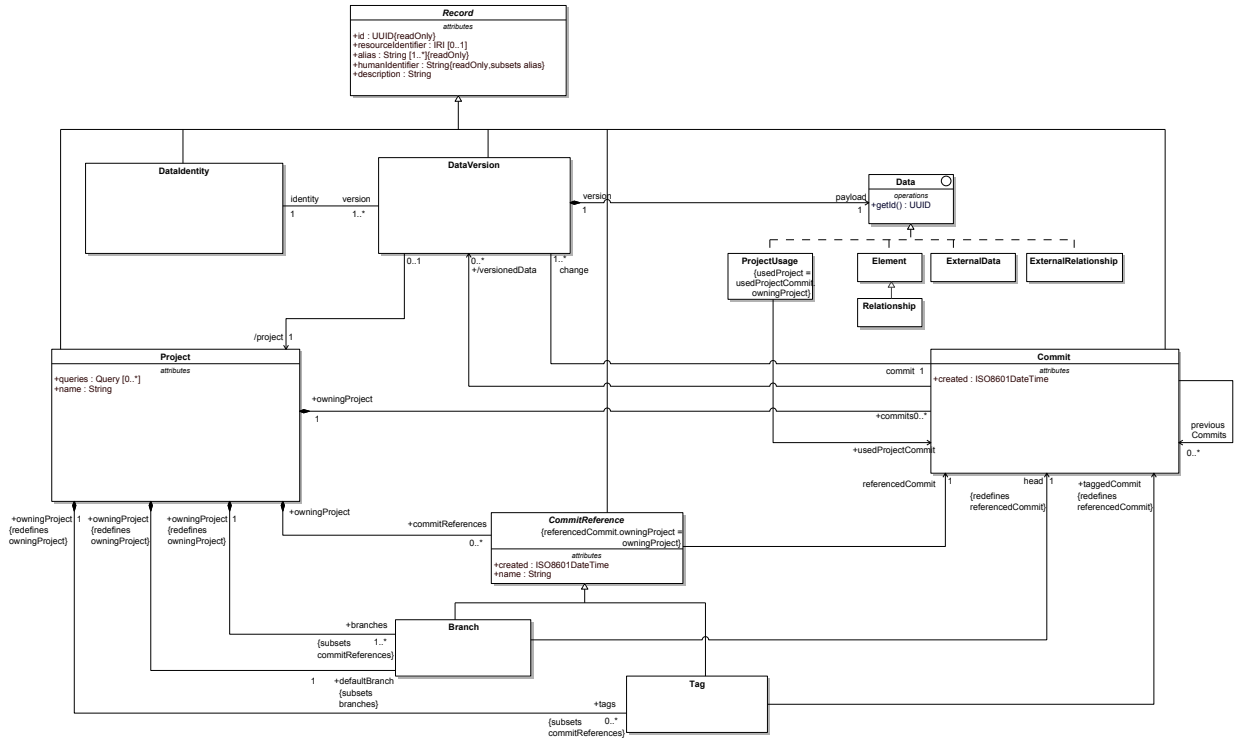

Figure 4. Project Data Versioning API Model

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]
- External Data
- External Relationship
- 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 attributes:

- *project* is a derived attribute that references the owning Project
- *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*: project commit at which the wrapped data (*payload*) was created, modified, or deleted.
- */project*: 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
- *identifiedData* is a derived attribute that is the set of DataIdentity records corresponding to the Data contained in the Project
- *commit* is the set of all commits in the Project
- *commitReference* is the set of all commit references in the Project
- *branch* is the set of all the branches in the Project and a subset of *commitReference*
- *defaultBranch* is the default branch in the Project and a subset of *branch*
- *tag* is the set of all the tags in the Project and a subset of *commitReference*
- *usage* is the set of ProjectUsage records representing all other Projects being used by the given Project (*ProjectUsage.usedProject)*
- *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.
- *previousCommit* 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 the set of cumulative DataVersion records in a Project at the 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[1]. 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 the normal end-user operations to preserve Project history.

[1]Note: A provider tool 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 (Commit Reference.*referencedCommit*). Project.*commit* is the set of all the Commit records for a given Project. Project.*commitRefererence* 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

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.previousCommit*, 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 immutable. 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 for Commit, Branch, and Tag.

Type of Record	Mutable	Destructible
Commit	No	No

Type of Record	Mutable	Destructible
Branch	Yes	Yes
Tag	No	Yes

7.1.3 ExternalData and ExternalRelationship

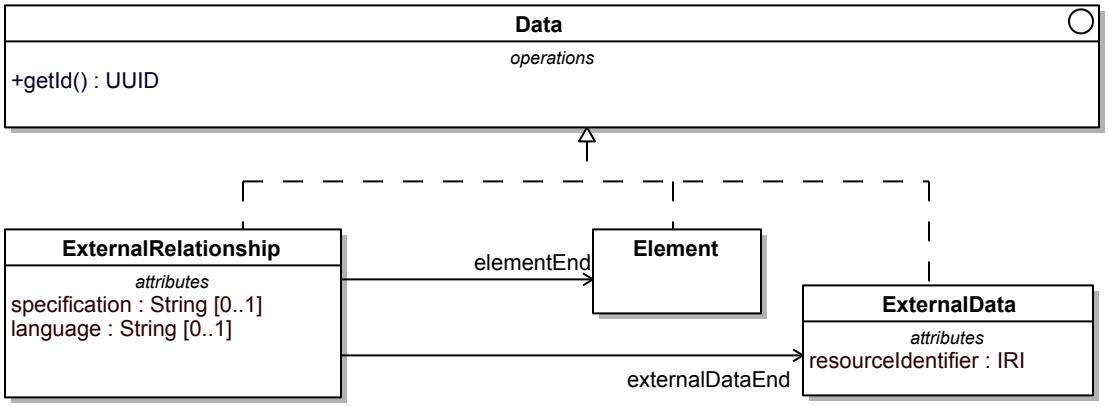

Figure 5. External Relationship API Model

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 external relationship. 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 the mathematical expressions (e.g. 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

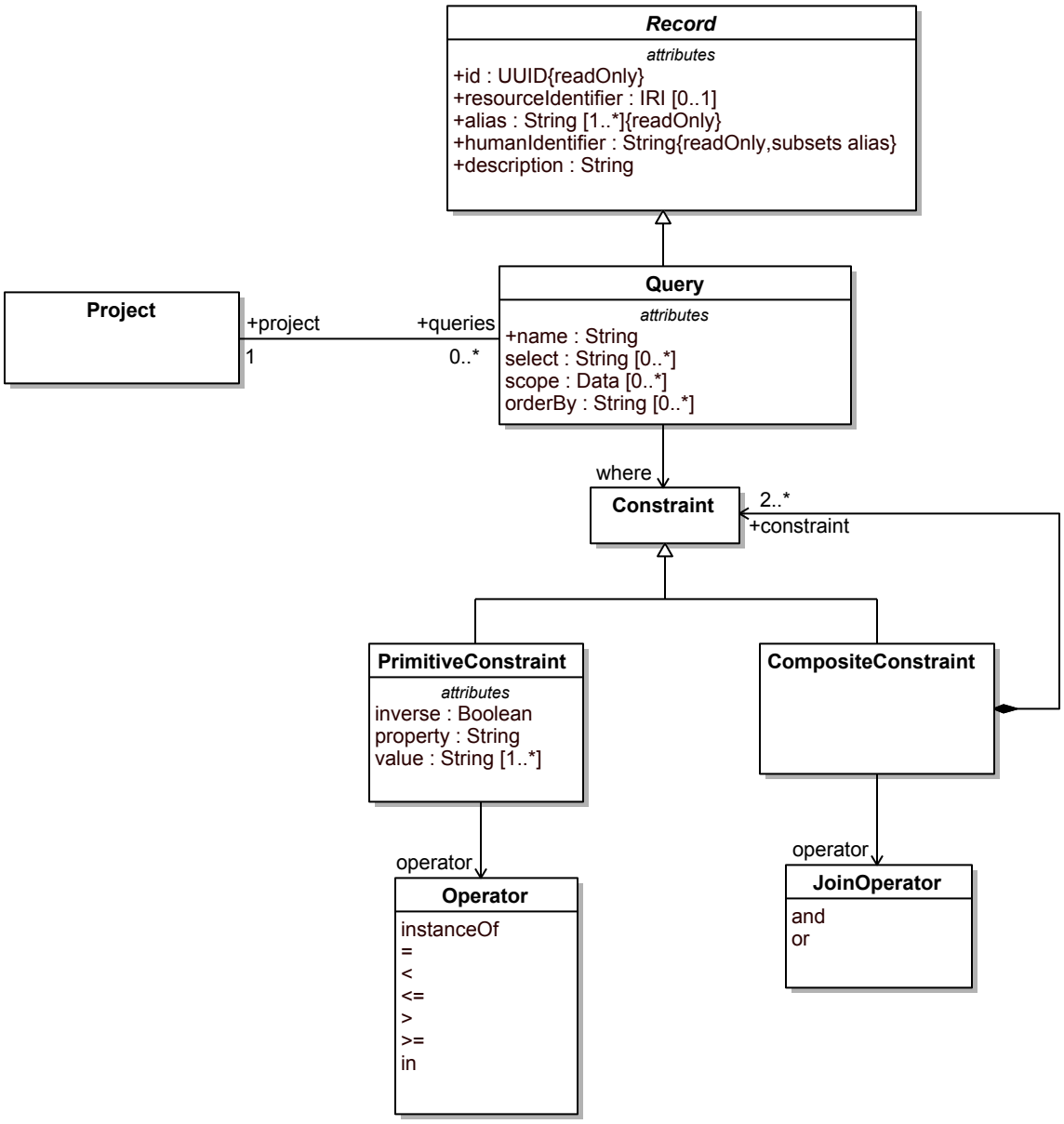

Figure 6. Query API Model

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 additional attributes:

- *name* is the name of the Query

- *select* is a list of properties of Element or its subtypes that will be included for each Element object in the query response. Element is the root-metaclass in KerML. If no properties are specified, then all the properties will be included for each Element in the query response.
- *scope* is a list of Element objects that define the scope for query execution. The default scope of a Query is the owning project.
- *where* is of type Constraint and represents the conditions that Elements in the query response must satisfy.
- *orderBy* is a list of properties of Element or its subtypes that are used for sorting the Element objects in the query response. The order of properties in the list governs the sorting order.

Constraint - Constraint represents conditions that must be satisfied by Element objects in the query response.

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

- *property* is a property of Element or its subtypes that is being constrained
- *operator* is of type Enumeration whose literals are mathematical operators, as listed 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 subtype of Constraint that represents complex conditions composed of two or more CompositeConstraints or PrimitiveConstraints using logical AND or OR operators. 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

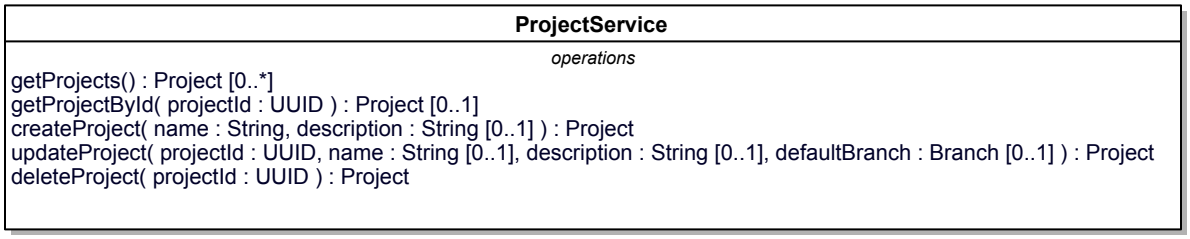

Figure 7. ProjectService Operations

Table 6. Operations

Name	Documentation
updateProject	Update the project with the given id (projectId).
getProjectById	Get project with the given id (projectId).
createProject	Create a new project with the given name and description (optional).
deleteProject	Delete the project with the given id (projectId).
getProjects	Get all projects.

7.2.2 ElementNavigationService

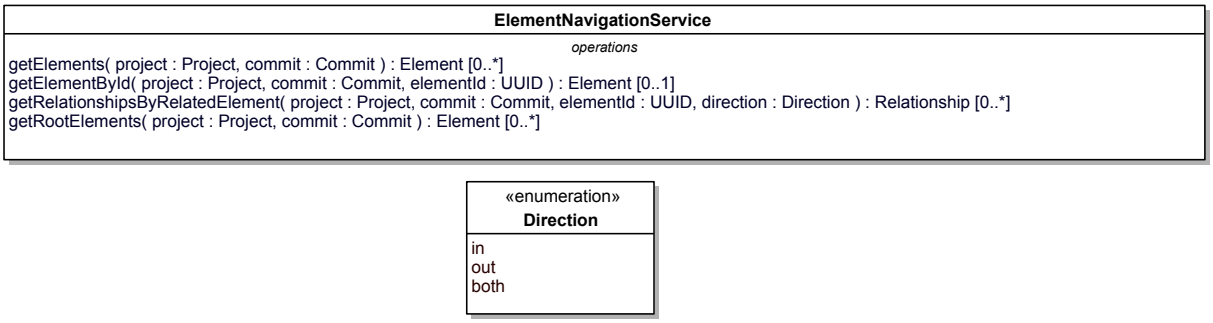

Figure 8. ElementNavigationService Operations

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 7. Operations

Name	Documentation
getElements	Get all the elements in a given project at the given commit.
getElementById	Get element with the given id (elementId) in the given project at the given commit.
getRelationshipsByRelatedElement	Get relationships that are incoming, outgoing, or both relative to the given related element.
getRootElements	Get all the root elements in the given project at the given commit.

7.2.3 ProjectDataVersioningService

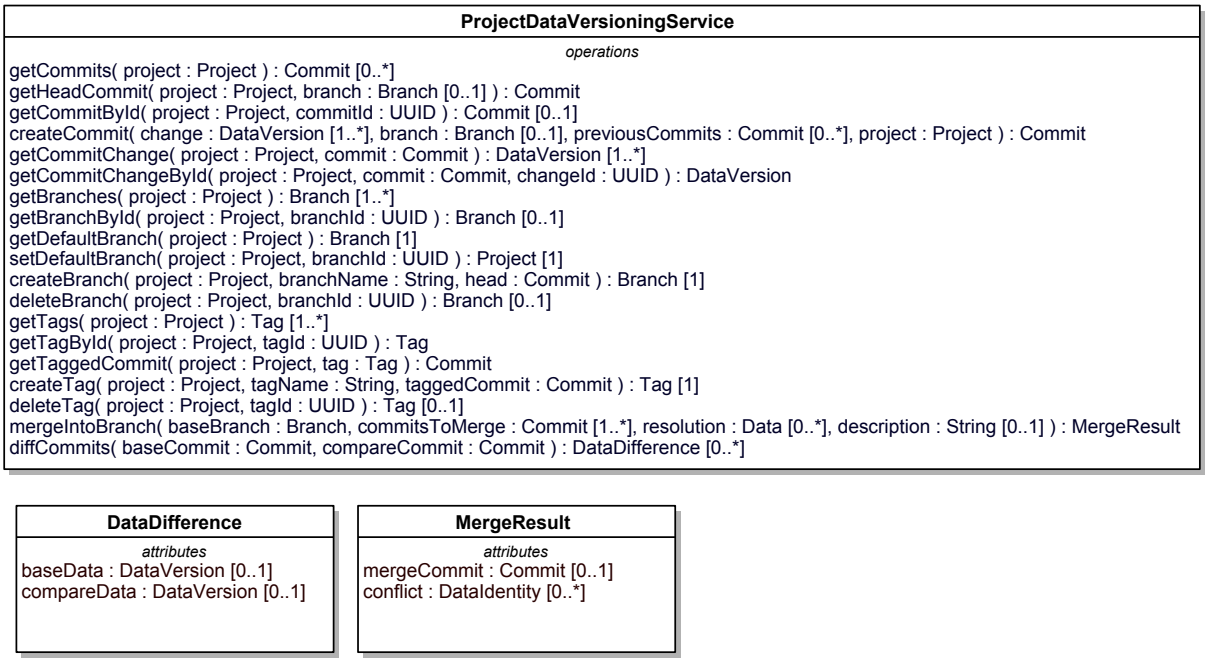

Figure 9. ProjectDataVersioningService Operations

Table 10. Operations

Name	Documentation
getCommitChangeById	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.
getBranchById	Get the branch with the given id (branchId) in the given project.
getTagById	Get the tag with the given id (tagId) in the given project.
getCommits	Get all the commits in the given project.
getBranches	Get all the branches in the given project.

Name	Documentation
createCommit	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.

Name	Documentation
diffCommits	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.
mergeIntoBranch	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.
getCommitById	Get the commit with the given id (commitId) in the given project.
deleteBranch	Delete the branch with the given id (branchId) in the given project.
getDefaultBranch	Get the default branch of the given project.
deleteTag	Delete the tag with the given id (tadId) in the given project.
setDefaultBranch	Set the branch with the given branchId as the default branch of the given project.
getHeadCommit	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.

Name	Documentation
getTaggedCommit	Get the tagged commit of the given tag in the given project.
getCommitChange	Get the change in the given commit of the given project.
getTags	Get all the tags in the given project.
createTag	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).
createBranch	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).

7.2.4 QueryService

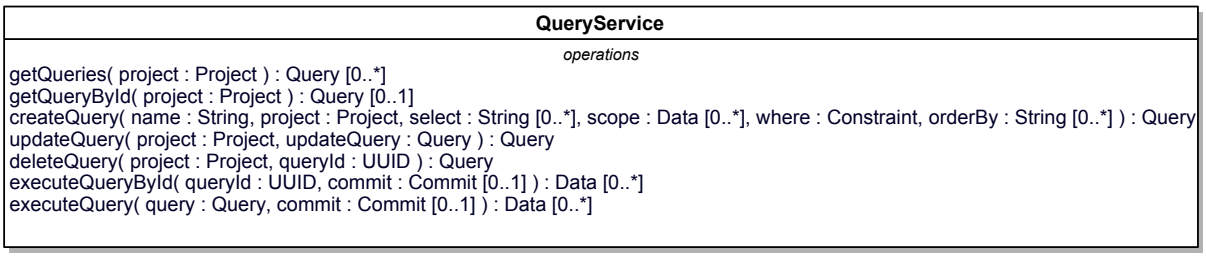

Figure 10. QueryService Operations

Table 11. Operations

Name	Documentation
getQueryById	Get the query with the given id (queryId) in the given project.
getQueries	Get all the queries in the given project.
executeQueryById	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.
createQuery	Create a query in the given project with the given inputs.
deleteQuery	Delete the query with the given id (queryId) in the given project.
updateQuery	Update the given query (updateQuery) in the given project.
executeQuery	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.

7.2.5 ExternalRelationshipService

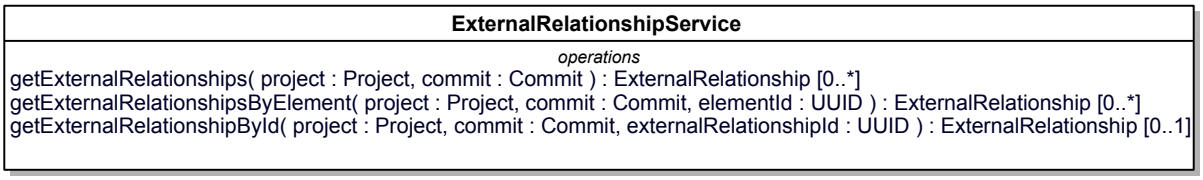

Figure 11. ExternalRelationshipService Operations

Table 12. Operations

Name	Documentation
getExternalRelationshipById	Get the external relationship with the given id (externalRelationshipId).
getExternalRelationshipsByElement	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.
getExternalRelationships	Get all the external relationships in a given project at a given commit.

7.2.6 ProjectUsageService

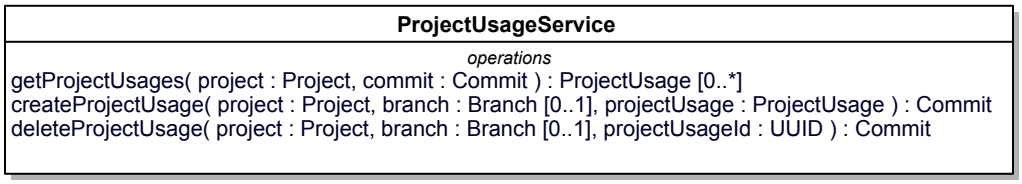

Figure 12. ProjectUsageService Operations

Table 13. Operations

Name	Documentation
createProjectUsage	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.
deleteProjectUsage	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.
getProjectUsages	Get all the project usages in the given project at the given commit.

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) 3.0 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 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 14. PIM API Model - REST/HTTP PSM Model Mapping Table

PIM Concept	REST/HTTP PSM Model (JSON)
Project	Project
Branch	Branch
Tag	Tag
Commit	Commit
Element	Element
Relationship	Relationship
Data	Data
Data Identity	DataIdentity
Data Version	DataVersion
Query	Query
Constraint	Constraint
Primitive Constraint	PrimitiveConstraint
Composite Constraint	CompositeConstraint

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 15. PIM to REST / HTTP PSM Mapping

PIM Service	REST / HTTP PSM Endpoint
ProjectService	

PIM Service	REST / HTTP PSM Endpoint
createProject	POST /projects
getProjects	GET /projects
getProjectById	GET /projects/{projectId}
updateProject	PUT /projects/{projectId}
deleteProject	DELETE /projects/{projectId}
ElementNavigationService	
getElements	GET/projects/{projectId}/commits/{commitId}/elements
getElementById	GET /projects/{projectId}/commits/{commitId}/elements/{elementId}
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
getCommitById	GET /projects/{projectId}/commits/{commitId}

PIM Service	REST / HTTP PSM Endpoint
createCommit	POST /projects/{projectId}/commit • The body of the POST request is a CommitRequest. • The content of CommitRequest.change for creating, updating, and deleting Data maps directly to the corresponding PIM operation (createCommit), and is described below. ◦ For creating new Data, CommitRequest.change should include a DataVersion where DataVersion.payload includes the Data being created and DataVersion.identity is not specified. ◦ 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. ◦ For deleting existing Data, CommitRequest.change should include a DataVersion where DataVerision.payload is not specified, and DataVersion.identity is the DataIdentity that will be deleted in the commit.
getCommitChange	GET /projects/{projectId}/commits/{commitId}/changes
getCommitChangeById	GET /projects/{projectId}/commits/{commitId}/changes/{changeId}
getBranches	GET /projects/{projectId}/branches
getBranchById	GET /projects/{projectId}/branches/{branchId}
getDefaultBranch	• GET /projects/{projectId} returns a Project with the given ID (projectId) • Project.*defaultBranch* provides the ID of default branch of the project
setDefaultBranch	• PUT /projects/{projectId} • The body of the PUT request conforms to the Project JSON model. Set the ID of the new default branch (Project.defaultBranch) in the body.
createBranch	POST /projects/{projectId}/branches
deleteBranch	DELETE /projects/{projectId}/branches/{branchId}
getTags	GET /projects/{projectId}/tags
getTagById	GET /projects/{projectId}/tags/{tagId}

PIM Service	REST / HTTP PSM Endpoint
getTaggedCommit	• GET /projects/{projectId}/tags/{tagId} returns the tag with the given ID (tagId) • Tag.taggedCommit provides the ID of the tagged commit • GET /projects/{projectId}/commits/{commitId} returns the tagged commit given its ID (see the previous step)
createTag	POST /projects/{projectId}/tags
deleteTag	DELETE /projects/{projectId}/tags/{tagId}
mergeIntoBranch	POST /projects/{projectId}/branches/{targetBranchId}/merge
diffCommits	GET /projects/{projectId}/commits/{compareCommitId}/diff
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	

getExternalRelationships	• POST /projects/{projectId}/queries with Query 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 inputs to get_external_relationships

PIM Service	REST / HTTP PSM Endpoint
getExternalRelationshipsByElement	• POST /projects/{projectId}/queries with Query JSON model ◦ Query.*where* is set to a CompositeConstraint ◦ CompositeConstraint.*constraints* includes the following 2 instances of PrimitiveConstraint with the *and* operator ▪ PrimitiveConstraint 1 ▪ PrimitiveConstraint.*property* = @type ▪ PrimitiveConstraint.*value* = 'ExternalRelationship' ▪ PrimitiveConstraint.*operator* = '=' ▪ PrimitiveConstraint 2 ▪ PrimitiveConstraint.*property* = elementEnd ▪ PrimitiveConstraint.*value* = {elementId} ▪ PrimitiveConstraint.*operator* = '=' • Execute the query with the following request ◦ GET /projects/{projectId}/queries/{queryId}/results?commitId={commitId}, where {projectId} and {commitId} are inputs to getExternalRelationships
getExternalRelationshipsById	• POST /projects/{projectId}/queries with Query JSON model ◦ Query.*where* is set to a CompositeConstraint ◦ CompositeConstraint.*constraints* includes the following 2 instances of PrimitiveConstraint with the *and* operator ▪ PrimitiveConstraint 1 ▪ PrimitiveConstraint.*property* = @type ▪ PrimitiveConstraint.*value* = 'ExternalRelationship' ▪ PrimitiveConstraint.*operator* = '=' ▪ PrimitiveConstraint 2 ▪ PrimitiveConstraint.*property* = @id ▪ PrimitiveConstraint.*value* = {externalRelationshipId} ▪ PrimitiveConstraint.*operator* = '=' • Execute the query with the following request ◦ GET /projects/{projectId}/queries/{queryId}/results?commitId={commitId}, where {projectId} and {commitId} are inputs to getExternalRelationships
Project Usage Service	

PIM Service	REST / HTTP PSM Endpoint
getProjectUsages	• POST /projects/{projectId}/queries with Query 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 Project and Commit input parameters in get_project_usages
createProjectUsage	• POST /projects/{projectId}/commits?branchId={branchId} with Commit JSON model, such that: ◦ Commit.*change* = DataVersion with the following inputs ▪ DataVersion.*data* = ProjectUsage with the following inputs ▪ ProjectUsage.*usedProjectCommit* = {commitId} of the Project and Commit being used.
deleteProjectUsage	• POST /projects/{projectId}/commits?branchId={branchId} with Commit JSON model, such that: ◦ Commit.*change* = DataVersion with the following inputs ▪ DataVersion.*identity* = DataIdentity with the following inputs ▪ DataIdentity.*id* = {projectUsageId} ▪ DataVersion.*data* = 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. *and 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.

```
<http://sysml2-api-host:9000/projects?
    page[after]=MTYxODg2MTQ5NjYzMnwyMDEwOWY0MC00ODI1LTQxNmEtODZmNi03NTA4YWM0MmEwMjE&
    page[size]=3>; rel="next",
<http://sysml2-api-host:9000/projects?
    page[before]=MTYxODg2MTQ5NjYzMnwxMDg2MDFjMS1iNzk1LTRkMGEtYTFiYy1lZjEyYmMwNTU5ZjI&
    page[size]=3>; rel="prev"
```

Example

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.

```
http://sysml2-api-host:9000/projects?page[size]=3
```

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.

```
<http://sysml2-api-host:9000/projects?
    page[after]=MTYxODg2MjE2NTMxNXwwOGY0MzNkYi1iNmQ0LTQxYjgtOTAyMC1lODIwZWJjNDE3YmU&
    page[size]=3>; rel="next"
```

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*.

```
<http://sysml2-dev.intercax.com:9000/projects?
    page[after]=MTYxODg2MjY4OTYxNHwyMDEwOWY0MC00ODI1LTQxNmEtODZmNi03NTA4YWM0MmEwMjE&
    page[size]=3>; rel="next",
<http://sysml2-dev.intercax.com:9000/projects?
    page[before]=MTYxODg2MjY4OTYxNHwxMDg2MDFjMS1iNzk1LTRkMGEtYTFiYy1lZjEyYmMwNTU5ZjI&
    page[size]=3>; rel="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/.

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 OSLC Configuration Management specification. This mapping is shown in the table below. Refer to the OSLC Configuration Management specification for the resource shapes and vocabulary for the resource types listed in the table below - see https://oslc-op.github.io/oslc-specs/specs/config/oslc-config-mgt.html.

Table 16. PIM Concept to OSLC Resource type Mapping

PIM Concept	OSLC Configuration Management Resource Type	`dcterms:identifier`
Project	Component (oslc_config:Component)	PIM project id
Branch	Stream (oslc_config:Stream)	PIM branch id
Tag	Baseline (oslc_config:Baseline)	PIM tag id
Commit	Not supported by OSLC Configuration Management.	PIM commit id
Data Identity	Concept resource	PIM Data Identity id
Data Version	Version resource (oslc_config:VersionResource)	PIM Data Version id

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 17. PIM API Services - OSLC Services Mapping

PIM Services	OSLC PSM
ProjectService	

getProjectById	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):
	
	1. Discover a query capability for components (oslc_config:Component). 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 components with identifier 123, returning all properties.
	
	`GET queryBaseUri?`
	`oslc.where=dcterms%3Aidentifier%3D%22123%22&`
	`oslc.select=*`

PIM Services	OSLC PSM
createProject	1. Discover a creation factory for components (oslc_config:Component). See *OSLC Discovery* section. 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 https://docs.oasis-open-projects.org/oslc-op/config/v1.0/ps01/config-resources.html#ComponentShape). Example: `POST creationFactoryUri`

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)
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. However, providers may implement deletion operation for Components.
ElementNavigationService	

PIM Services	OSLC PSM

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`

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=*`

PIM Services	OSLC PSM

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
	
	Example1: 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>`
	
	Example2: Query for Relationships incoming to an element <http://sysml2.server.com/elements/123>:
	
	`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.

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.
ProjectDataVersioningService	

getBranchById	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=*`

PIM Services	OSLC PSM

getTagById	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=*`

getCommits	Not available. The OSLC Configuration Management specification does not define any notion of a commit. Versions of specific resources may be fetched.

getBranches	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`

PIM Services	OSLC PSM

createCommit	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.
	
	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.
	Example 1:
	
	`PUT conceptResourceUri?`
	`oslc_config.context=urlEncodedStreamUri`
	
	Example 2:
	
	`Headers: Configuration-Context=streamUri`
	`PUT conceptResourceUri`

getCommitById	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.
	
	`GET queryBaseUri?`
	`oslc.where=dcterms%3Aidentifier%3D%22123%22&`
	`oslc.select=*`

getCommitChange	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.

PIM Services	OSLC PSM
getCommitChangeById	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.
deleteBranch	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 "true"^^xsd:boolean property.
getDefaultBranch	Not available. The OSLC Configuration Management specification does not define any notion of a default stream.
deleteTag	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 "true"^^xsd:boolean property.
setDefaultBranch	Not available. The OSLC Configuration Management specification does not define any notion of a default stream.
getHeadCommit	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.
getTaggedCommit	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.

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`

PIM Services	OSLC PSM
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) 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)
	
	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)
	
	Example:
	
	`POST streamsLdpcUri`

mergeIntoBranch	Not Available
diffCommits	Not Available
QueryService	
getQueryById	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.

PIM Services	OSLC PSM
getQueries	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.
executeQueryById	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.
createQuery	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.
updateQuery	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.
deleteQuery	Not Available OSLC does not provide any RDF representation or persistence mechanisms of OSLC queries.

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.

ExternalRelationshipService	OSLC does/can not differentiate between properties/relationships/elements that are external or otherwise. The services below are just specific examples of the get_elements; get_element_by_id services above.
getExternalRelationshipsById	See getElementById under ElementNavigationService
getExternalRelationshipsByElementEnd	See getRelationshipsByRelatedElement under ElementNavigationService
getExternalRelationships	See getElements under Element Navigation Service
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.
getProjectUsages	Not available. The OSLC Configuration Management specification does not define any notion of component usage.

A Annex: Conformance Test Suite

(Normative)

A.1 ProjectService Conformance Test Cases

Operation create_project

PIM-PS-001

Description	Create Project - success
Input	1. `_description : String[0..1]` 2. `_name : String[1]`
Scenario	
Precondition (OCL)	`let _record : Record = Record.allInstances()`
Steps	Execute operation `create_project(name = _name, description = _description) : Project[1]`
Result	1. Result, defined as `project : Project[1]`, is a created Project 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 3. A Branch is created and specified as the `defaultBranch` attribute value for the created Project 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. 5. The `name` attribute value of the created Branch (`defaultBranch`) is specified as `'main'` 6. The `description` attribute value is specified as inputted 7. The `name` attribute value is specified as inputted

Postcondition (OCL)	`let project : Project = create_project(_name, _description)`
	`project->size() = 1`
	`Project.allInstances()->includes(project)`
	`_record.id->excludes(project.id)`
	`project.defaultBranch->notEmpty()`
	`_record.id->excludes(project.defaultBranch.id)`
	`project.defaultBranch.name = 'main'`
	`project.description = _description`
	`project.name = _name`

Operation get_projects

PIM-PS-002

Description	Get Projects - success
Input	None
Scenario	
Precondition (OCL)	
Steps	Execute operation `get_projects() : Project[0..*]`
Result	Result, defined as `project : Project[0..*]`, is all of the existing Projects
Postcondition (OCL)	`let project : Project = get_projects()` `project = Project.allInstances()`

Operation get_project_by_id

PIM-PS-003

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

PIM-PS-004

Description	Get Project by ID - does not exist
Input	1. `_projectId : UUID[1]`
Scenario	A Project with an `id` attribute value equal to `_projectId` does not exist
Precondition (OCL)	`Project.allInstances()->select(projectId = _projectId)->isEmpty()`

Steps	Execute operation `get_project_by_id(projectId = _projectId) : Project[0..1]`
Result	1. Result, defined as `project : Project[0]`, does not include any Projects 2. Result communicates that a Project with the provided ID does not exist
Postcondition (OCL)	`let project : Project = get_project_by_id(_projectId)` `project->isEmpty()`

A.2 ElementNavigationService Conformance Test Cases

Operation get_elements

PIM-EN-001

Description	Get Elements - success
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)	`Project.allInstances()->includes(_project)`
	`Commit.allInstance()->includes(_commit)`
	`_commit.owningProject = _project`

Steps	Execute operation `get_elements(project = _project, commit = _commit) : Element[0..*]`
Result	Result, defined as `element : Element[0..*]`, is all Elements at the inputted Commit
Postcondition (OCL)	`let element : Element = get_elements(_project, _commit)` `element = _commit.version.data->select(oclIsKindOf(Element))`

Operation get_element_by_id

PIM-EN-002

Description	Get Element by ID - success
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()->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]`

Result	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)` `element->isEmpty()`

Operation get_relationships_by_source

PIM-EN-004

Description	Get Relationships by source (Element) - success
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()->includes(_project)`
	`Commit.allInstance()->includes(_commit)`
	`_commit.owningProject = _project`
	`let _element : Element = _commit.version->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->select(data.oclIsKindOf(Relationship) and data.source->includes(_element))`

Operation get_relationships_by_target

PIM-EN-005

Description	Get Relationships by target (Element) - success

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()->includes(_project)`
	`Commit.allInstance()->includes(_commit)`
	`_commit.owningProject = _project`
	`let _element : Element = _commit.version->select(identity.id = _id and data.oclIsKindOf(Element)).data`
	`_element.size() = 1`

Steps	Execute operation `get_relationships_by_target(project = _project, commit = _commit, elementId = _elementId) : Relationship[0..*]`
Result	Result, defined as `relationship : Relationship[0..*]`, is all the Relationships whose `target` attribute value includes the Element with `id` attribute value equal to `_elementId`
Postcondition (OCL)	`let relationship : Relationship = get_relationship_by_target(_project, _commit, _elementId)` `relationship = _commit.version->select(data.oclIsKindOf(Relationship) and data.target->includes(_element))`

A.3 ProjectDataVersioningService Conformance Test Cases

Operation create_branch

PIM-PCB-001

Description	Create Branch - success
Input	1. `_project : Project[1]` 2. `_head : Commit[1]` 3. `_name : String[1]`
Scenario	1. The inputted Project exists 2. The inputted Commit (`_head`) exists 3. The inputted Commit (`_head`) belongs to the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`Commit.allInstance()->includes(_head)`
	`_head.owningProject = _project`
	`let _record : Record = Record.allInstances()`

Steps	Execute operation `create_branch(project = _project, head = _head, name = _name) : Branch[1]`
Result	1. Result, defined as `branch : Branch[1]`, is a created Branch in the inputted Project 2. The `id` attribute value of the created Branch 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 Branch is specified as inputted, i.e. equal to `_project` 4. The `head` attribute value of the created Branch is specified as inputted 5. The `name` attribute value of the created Branch is specified as inputted

Postcondition (OCL)	`let branch : Branch = create_branch(_project, _head, _name)`
	`branch->size() = 1`
	`Branch.allInstances()->includes(branch)`
	`_project.branch->includes(branch)`
	`_record.id->excludes(branch.id)`
	`branch.owningProject = _project`
	`branch.head = _head`
	`branch.name = _name`

Operation get_branches

PIM-PCB-002

Description	Get Branches - success
Input	1. `_project : Project[1]`
Scenario	The inputted Project exists
Precondition (OCL)	`Project.allInstances()->includes(_project)`
Steps	Execute operation `get_branches(project = _project) : Branch[0..*]`
Result	Result, defined as `branch : Branch[0..*]`, is all Branches in the inputted Project
Postcondition (OCL)	`let branch : Branch = get_branches(_project)` `branch = _project.branch`

Operation get_branch_by_id

PIM-PCB-003

Description	Get Branch by ID - success
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` exists in the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`let _branch : Branch = _project.branch->select(id = _branchId)`
	`_branch.size() = 1`

Steps	Execute operation `get_branch_by_id(project = _project, branchId = _branchId) : Branch[0..1]`
Result	Result, defined as `branch : Branch[1]`, is the Branch with an `id` attribute value equal to `_branchId` in the inputted Project
Postcondition (OCL)	`let branch : Branch = get_branch_by_id(_project, _branchId)` `branch = _branch`

PIM-PCB-004

Description	Get Branch by ID - 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()`
Steps	Execute operation `get_branch_by_id(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 = get_branch_by_id(_project, _branchId)` `branch->isEmpty()`

Operation delete_branch

PIM-PCB-005

Description	Delete Branch - success
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` exists in the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`let _branch : Branch = _project.branch->select(id = _branchId)`
	`_branch.size() = 1`

Steps	Execute operation `delete_branch(project = _project, branchId = _branchId) : Branch[0..1]`
Result	1. Result, defined as `branch : Branch[1]`, is the Branch with an `id` attribute value equal to `_branchId` in the inputted Project 2. Branch with an `id` attribute value equal to `_branchId` does not exist

Postcondition (OCL)	`let branch : Branch = delete_branch(_project, _branchId)`
	`_project.branch->excludes(branch)`
	`Branch.allInstances()->excludes(branch)`
	`_project.branch->select(id = _branchId)->isEmpty()`
	`Branch.allInstances()->select(id = _branchId)->isEmpty()`

PIM-PCB-006

Description	Delete 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()`
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

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` exists in the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`let _branch : Branch = _project.branch->select(id = _branchId)`
	`_branch.size() = 1`

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 = _branch`

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 `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)	`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)	`Project.allInstances()->includes(_project)`
	`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)	`let commit : Commit = get_commit_by_id(_project, _commitId)` `commit = _commit`

PIM-PCB-012

Description	Get commit by ID - does not exist in Project
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` does not exist in the inputted Project
Precondition (OCL)	`Project.allInstances()->includes(_project)` `_project.commit->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)	`let commit : Commit = get_commit_by_id(_project, _commitId)` `commit->isEmpty()`

Operation get_head

PIM-PCB-013

Description	Get head Commit of Branch - success
Input	1. `_project : Project[1]` 2. `_branch : Branch[1]`
Scenario	1. The inputted Project exists 2. The inputted Branch exists 3. The inputted Branch belongs to the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`Branch.allInstance()->includes(_branch)`
	`_branch.owningProject = _project`

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

Result	Result, defined as `head : Commit[0..1]`, is the Commit that is the `head` attribute value of the inputted Branch

Postcondition (OCL)	`let head : Commit = get_head(_project, _branch)`
	`Bag{0, 1}->includes(head->size())`
	`head = _project.head`

PIM-PCB-014

Description	Get head Commit of Branch - Branch does not exist
Input	1. `_project : Project[1]` 2. `_branch : Branch[1]`
Scenario	1. The inputted Project exists 2. The inputted Branch does not exist
Precondition (OCL)	`Project.allInstances()->includes(_project)` `Branch.allInstance()->excludes(_branch)`
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 inputted Branch does not exist
Postcondition (OCL)	`let head : Commit = get_head(_project, _branch)` `head->isEmpty()`

PIM-PCB-015

Description	Get head Commit of Branch - Branch not in Project
Input	1. `_project : Project[1]` 2. `_branch : 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()->includes(_project)`
	`Branch.allInstance()->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

Description	Create Query - success
Input	1. `_project : Project[1]` 2. `_select : String[0..*]` 3. `_scope : DataIdentity[0..*]` 4. `_where : Constraint[0..1]` 5. `_orderBy : String[0..*]`
Scenario	1. The inputted Project exists
Precondition (OCL)	`Project.allInstances()->includes(_project)` `let _record : Record = Record.allInstances()`
Steps	Execute operation `create_query(project = _project, select = _select, scope = _scope, where = _where, orderBy = _orderBy) : Query[1]`

Result	1. Result, defined as `query : Query[1]`, is a created Query in the inputted Project 2. The `id` attribute value of the created Query 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 Query is specified as inputted, i.e. equal to `_project` 4. The `select` attribute value of the created Query is specified as inputted 5. The `scope` attribute value of the created Query is specified as inputted 6. The `where` attribute value of the created Query is specified as inputted 7. The `orderBy` attribute value of the created Query is specified as inputted

Postcondition (OCL)	`let query : Query = create_query(_project, _select, _scope, _where, _orderBy)`
	`query->size() = 1`
	`Query.allInstances()->includes(query)`
	`_project.query->includes(query)`
	`_record.id->excludes(query.id)`
	`query.owningProject = _project`
	`query.select = _select`
	`query.scope = _scope`
	`query.where = _where`
	`query.orderBy = orderBy`

PIM-QS-002

Description	Execute Query - success
Input	1. `_query : Query[1]` 2. `_commit : Commit[1]`
Scenario	1. The inputted Query exists 2. The inputted Commit exists 3. The inputted Query and Commit both belong to the same Project

Precondition (OCL)	`Query.allInstances()->includes(_query)`
	`Commit.allInstances()->includes(_commit)`
	`_query.owningProject = _commit.owningProject`

Steps	Execute operation `execute_query(query = _query, commit = _commit) : Data[0..*]`
Result	1. Result, defined as `data : Data[0..*]`, is all of the Datas at the inputted Commit that satisfy the conditions of the inputted Query 2. If inputted Query has a non-empty `scope` attribute value, result only includes Data within the Query's scope

Postcondition (OCL)	`let data : Data = execute_query(_query, _commit)`
	`_commit.versionedData->includesAll(data)`
	`_query.scope->isEmpty() or _query.scope->includesAll(data)`

A.5 ExternalRelationshipService Test Cases

Operation get_external_relationships

PIM-ER-001

Description	Get ExternalRelationships - success
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)	`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)	`let externalRelationship : ExternalRelationship = get_external_relationships(_project, _commit)` `externalRelationship = _commit.version.data->select(oclIsKindOf(ExternalRelationship))`

Operation get_external_relationship_by_id

PIM-ER-002

Description	Get ExternalRelationship by ID - success
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)	`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)	`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

Description	Get ExternalRelationships - success
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)	`Project.allInstances()->includes(_project)`
	`Commit.allInstance()->includes(_commit)`
	`_commit.owningProject = _project`

Steps	Execute operation `get_project_usages(project = _project, commit = _commit) : ProjectUsage[0..*]`

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

Description	Execute operation - missing Project input
Input	1. `_project : Project[0]`
Scenario	
Precondition (OCL)	
Steps	Execute any of the following operations, defined as `x(project : Project[1], …)`: • `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

Description	Execute operation - missing Commit input
Input	1. `_commit : Commit[0]`
Scenario	
Precondition (OCL)	
Steps	Execute any of the following operations, defined as `x(commit : Commit[1], …)`: • `get_elements(…, commit = _commit)` • `get_element_by_id(…, commit = _commit, …)` • `get_relationships_by_source(…, commit = _commit, …)` • `get_relationships_by_target(…, commit = _commit, …)` • `create_branch(…, head = _commit, …)`
Result	1. Result, defined as `result : OclAny[0]`, does not include any OclAny 2. Result communicates that `commit` is a required input
Postcondition (OCL)	`let result : OclAny = x(_commit, …)` `result->isEmpty()`

PIM-CC-003

Description	Execute operation - Project input does not exist
Input	1. `_project : Project[1]`
Scenario	The inputted Project does not exist
Precondition (OCL)	`Project.allInstances()->excludes(_project)`

Steps	Execute any of the following operations, defined as `x(project : Project[1], …)`: • `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 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], …)`: • `get_elements(…, commit = _commit)` • `get_element_by_id(…, commit = _commit, …)` • `get_relationships_by_source(…, commit = _commit, …)` • `get_relationships_by_target(…, commit = _commit, …)` • `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

Postcondition (OCL)	`let result : OclAny = x(_commit, …)` `result->isEmpty()`

PIM-CC-005

Description	Execute operation - Commit input is not owned by Project input
Input	1. `_project : Project[1]` 2. `_commit : Commit[1]`
Scenario	1. The inputted Project exists 2. The inputted Commit does not exist 3. The inputted Commit does not belong to the inputted Project

Precondition (OCL)	`Project.allInstances()->includes(_project)`
	`Commit.allInstances()->includes(_commit)`
	`_commit.owningProject <> _project`

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, …)`
Result	1. Result, defined as `result : OclAny[0]`, does not include any OclAny 2. Result communicates that the provided input is invalid
Postcondition (OCL)	`let result : OclAny = x(_project, _commit, …)` `result->isEmpty()`

PIM-CC-006

Description	Execute operation - missing name input
Input	1. `_name : String[0]`
Scenario	

Precondition (OCL)	
Steps	Execute any of the following operations, defined as `x(name : String[1], …) : OclAny`: • `create_project(name = _name, …)` • `create_branch(…, name = _name, …)`
Result	1. Result, defined as `result : OclAny[0]`, does not include any OclAny 2. Result communicates that `name` is a required input
Postcondition (OCL)	`let result : OclAny = x(_name, …)` `result->isEmpty()`

PIM-CC-007

Description	Execute operation - missing UUID input
Input	1. `_id : UUID[0]`
Scenario	
Precondition (OCL)	
Steps	Execute any of the following operations, defined as `x(id : UUID[1], …) : OclAny`: • `get_project_by_id(projectId = _id)` • `get_element_by_id(…, elementId = _id)` • `get_relationships_by_source(…, elementId = _id)` • `get_relationships_by_target(…, elementId = _id)` • `get_branch_by_id(…, branchId = _id)` • `delete_branch(…, branchId = _id)` • `set_default_branch(…, branchId = _id)` • `get_commit_by_id(…, commitId = _id)`
Result	1. Result, defined as `result : OclAny[0]`, does not include any OclAny 2. Result communicates that the input for which `_id` is used is a required input
Postcondition (OCL)	`let result : OclAny = x(_id, …)` `result->isEmpty()`

B Annex: API and Services Examples and Cookbook

(Informative)

B.1 Examples

In this subclause, 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 JSON response is present.

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 tool, 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:


```
[
  {
    "@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,
    "name": "8-Requirements Wed Oct 20 14:24:05 CDT 2021"
```

```
  }
]
```

GET /projects/{projectId}

Gets the project with id {projectId}.

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": "0a021fc1-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:

```
curl -X POST "Protocol://FQDN/projects" -H  "accept: application/json" -H
"Content-Type: application/json" -d "{  \"@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-107dc862c1ff"
  },
  "description": "Getting started with SysML v2 REST/HTTP API",
  "name": "SysML v2 API Start"
}
```

Branch

GET /projects/{projectId}/branches

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

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": "6b5e80d1-7291-4eef-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": "6b5e80d1-7291-4eef-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/6b5e80d1-7291-4eef-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": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "referencedCommit": {
    "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
  },
```

```
  "timestamp": "2021-11-06T15:45:34.967737-04:00"
}
```

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/6b5e80d1-7291-4eef-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\":
\"6b5e80d1-7291-4eef-880c-c462ca5a3288\"  }}"
```

Response (body):

```
{
  "@id": "feefb263-f330-49da-b679-1d65602bf611",
  "@type": "Branch",
  "head": {
    "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
  },
  "name": null,
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-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/6b5e80d1-7291-4eef-880c-c462ca5a3288/tags" -H
"accept: application/json"
```

Response:

```
[
  {
    "@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"
  }
]
```

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"
}
```

GET /projects/{projectId}/tags/{tagId}

Gets the tag with id {tagId} in the project with id {projectId}.

Request:

```
curl -X GET "Protocol://FQDN/projects/6b5e80d1-7291-4eef-880c-c462ca5a3288/tags/
d4657183-c4c1-45e9-90ac-dc984fc7c488" -H  "accept: application/json"
```

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": {
```

```
        "@id": "7787bf4a-48fb-4176-a495-d23b38c2f3ec",
        "@type": "DataIdentity"
      },
      "payload": null
    }
  ],
  "owningProject": {
    "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
  },
  "previousCommit": {
    "@id": "068440fd-9294-4acf-bc61-00b9156c9dde"
  },
  "timestamp": "2021-11-06T16:39:46.579269-04:00"
}
```


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",
```

```
    "owningProject": {
      "@id": "6b5e80d1-7291-4eef-880c-c462ca5a3288"
    },
    "scope": [],
    "select": [
      "owner",
      "@type",
      "name",
      "@id"
    ],
    "where": {
      "@type": "PrimitiveConstraint",
      "inverse": false,
      "operator": "=",
      "property": "name",
      "value": "Vehicle_A"
    }
  }
]
```

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\": \"@type\",    \"value\": \"PartDefinition\"  }}"
```

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}

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 (see also https://jupyter.org).

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