

**SECOM Co., Ltd. | VTT Building and Transport**  
**Yoshinobu Adachi**  
**E-Mail: yoshinobu.adachi@vtt.fi**

## VTT-TEC-ADA-12

# Overview of Partial Model Query Language

2002/05/14

1. INTRODUCTION .....	3
1.1 BACKGROUND OF PMQL .....	3
1.2 WHO IS PMQL FOR? .....	3
2. PMQL OVERVIEW .....	4
2.1 OBJECTIVE .....	4
2.2 WHAT DOES PMQL DO? .....	4
2.3 CONCEPT OF PMQL .....	4
3. PMQL ELEMENTS .....	5
3.1 <PMQL> .....	5
3.1.1 Format .....	5
3.1.2 Position .....	5
3.1.3 name Attribute .....	5
3.1.4 output Attribute .....	5
3.2 <SELECT> .....	5
3.2.1 Format .....	5
3.2.2 Position .....	5
3.2.3 type Attribute .....	5
3.2.4 match Attribute .....	6
3.2.5 action Attribute .....	6
3.3 <WHERE> .....	6
3.3.1 Format .....	6
3.3.2 Position .....	6
3.3.3 op Attribute .....	6
3.4 <OIDIS> .....	6
3.4.1 Format .....	6
3.4.2 Position .....	6
3.4.3 value Attribute .....	6
3.4.4 Example .....	7
3.5 <TYPEIS> .....	7
3.5.1 Format .....	7
3.5.2 Position .....	7
3.5.3 value Attribute .....	7
3.5.4 Example .....	7
3.6 <EXPR> .....	7
3.6.1 Format .....	7
3.6.2 Position .....	7
3.6.3 value Attribute .....	7
3.6.4 Example .....	7
3.7 <CASCADES> .....	8
3.7.1 Format .....	8
3.7.2 Position .....	8

3.7.3.	Example .....	8
3.8	<UPDATE> .....	8
3.8.1.	Format.....	8
3.8.2.	Position .....	8
3.8.3.	Example .....	8
4.	WHERE ELEMENT .....	9
4.1	BOOLEAN OPERATION BETWEEN WHERE ELEMENTS .....	9
4.2	BOOLEAN OPERATION BETWEEN ODIS ELEMENTS .....	9
4.3	BOOLEAN OPERATION BETWEEN TYPEIS ELEMENTS .....	9
4.4	BOOLEAN OPERATION BETWEEN EXPR ELEMENTS .....	10
5.	PMQL EXAMPLES .....	11
5.1	SELECT BY ENTITY TYPE .....	11
5.2	SELECT WITH WHERE RULE.....	11
5.3	SELECT WITH OID.....	11
5.4	SELECT WITH INVERSE .....	11
5.5	USING PREDEFINED PMQL.....	12

# 1. INTRODUCTION

This document describes the basic idea and technical overview of Partial Model Query Language, PMQL.

The partial model data exchanging has been a hot topic for some time, however there were no general and flexible means to describe the query information to handle the partial model data. There has been a lot of interactive user interface on the IFC compliant applications to select particular partial model, for example, using graphical user interface to set some specific values to select door objects that have less than 200 cm height, and so on. When you want to do this kind of query operation to model server, there seems to need a programmable query language for the partial model handling, in order to avoid human interactions.

PMQL enables to make flexible partial model query information by XML. PMQL itself is XML data, therefore it can be used in the transport of messages between Web Services using SOAP.

PMQL also potentially allows you to define 'view definition' data that describes which part of model data is required for the implementation with the specific view.

## 1.1 Background of PMQL

In the last few years, IFC related pilot projects are increasing in AEC/FM industry. By such industrial experiences, the end users of IFC compliant applications found that file based information exchange is not sufficient for the real world needs. There are several clear reasons for that as follows:

- File size would be huge, i.e. much larger than a few 10 Mbytes.
- Partial model data handling is difficult with the file exchange.
- Incremental model data management is difficult with file exchange.
- Keeping consistency is difficult by file exchange.

To improve these points, model server functionality has been focused as new communication tool between IFC compliant applications. The model server provides basic IFC model data import and export functionalities and also includes partial model selecting and updating functionalities as well.

When multiple different kind of client applications access to the model server, there might be multiple different requirements for partial model selection. For example, application A needs to select particular building element objects and its related geometric objects, however application B would try to select same building objects and its related property set objects without geometric objects. The model server must cover such different query requests from unspecific type of application, therefore hard-coded partial model query algorithm should be avoided to keep flexibility and extensibility. There were several requirements to solve this issue as follows:

- It can describe general partial model query definition.
- It can allow you to describe conditional expression for selection.
- It can be used in SOAP communication on the Internet.

PMQL was designed to realize these requirements.

## 1.2 Who is PMQL for?

PMQL is aimed at end users and programmers of IFC client software who want to use IFC Model Server. The IFC Model Server is the first implementation of PMQL interpreter.

## 2. PMQL Overview

### 2.1 Objective

The PMQL aims to provide a general means for select, update, and delete partial model data that contains specific part of product model data.

The PMQL provides a flexible and programmable partial model query to the client side application developers.

### 2.2 What does PMQL do?

The PMQL is an XML language to describe product model operations as follows:

- Select
- Update
- Delete

The PMQL describes the partial model object structures by XML element structures. To make a partial model query, the PMQL contains recursive object structures and conditional expression that is based on SQL. The object structure of PQML covers normal object reference and inverse relationship. The data formats of partial model that is returned by IFC Model Server are Enhanced BLIS-XML, BLIS-XML, and STEP Part 21 file format, ifcXML format would be covered in future release.

### 2.3 Concept of PMQL

The PMQL describes the partial model matching pattern directly by simple syntax. For example, the PMQL written in Figure 1 matches and returns the partial model data in Figure 2.

```

<pmql>
  <select type="entity" match="IfcWall" action="get">
    <where>
      <expr value="Label LIKE 'Wall#%'" />
      <expr value="calcWallVolume > 7.0" />
    </where>
    <cascades>
      <select type="attribute" match="LocalPlacement" action="get">
        <cascades>
          <select type="attribute" match="PlacementRelTo" action="get" />
          <select type="attribute" match="RelativePlacement" action="get">
            <cascades>
              <select type="attribute" match="Location" action="get" />
              <select type="attribute" match="Axis" action="get" />
              <select type="attribute" match="RefDirection" action="get" />
            </cascades>
          </select>
        </cascades>
      </select>
    </cascades>
  </select>
</pmql>

```

Figure 1. Example of PMQL

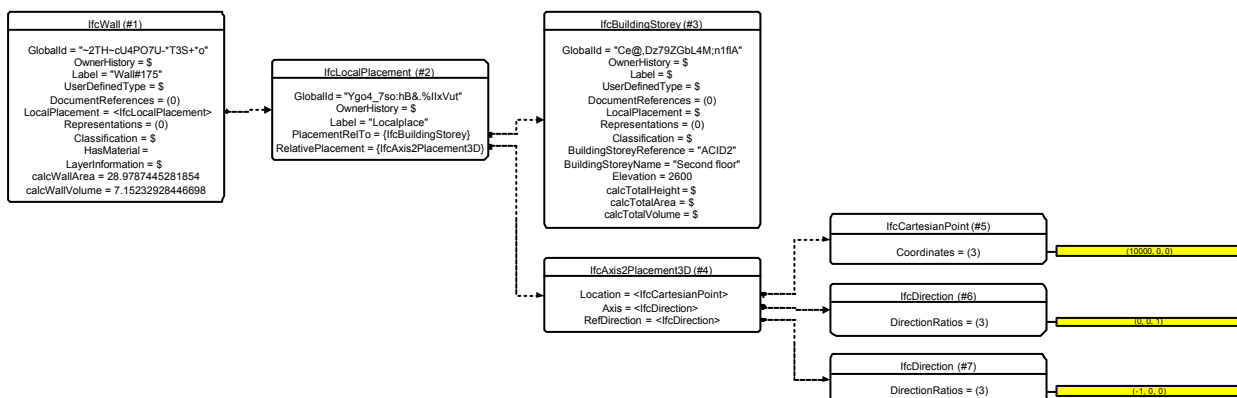


Figure 2. Result of PMQL

## 3. PMQL Elements

### 3.1 <pmql>

This element is the root element of PMQL data.

#### 3.1.1. Format

```
<pmql>
  pmql body
</pmql>
```

#### 3.1.2. Position

This element appears as a root element.

#### 3.1.3. name Attribute

This attribute defines the name of PMQL data. The name is based on URN (Uniform Resource Name). The URN of PMQL follows following convention (by order):

- urn: prefix of urn. This must be there at the beginning of URN.
- model-server: The identifier of model server.
- imsvr: The identifier of IFC Model Server.
- schema-version: The schema version. (equivalent with P21's schema identifier)
- entity-type: The entity type that is selected as mainly.
- view-definition –name or concept-name: The name of view definition.
- user-defined-method-name: The user defined methods for update, delete, select.

For example, “urn:model-server:imsvr:ifc20\_longform:ifcwall:geometry”

#### 3.1.4. output Attribute

This attribute defines the output format from IFC Model Server. The default format of returned IFC model data is EBLIS-XML. The output attribute takes following format types:

- eblis: Enhanced BLIS-XML format (default)
- blis: BLIS-XML format
- p21: STEP P21 format
- p28: STEP P28 format
- ifcxml: ifcXML format

## 3.2 <select>

This element is used to describe selection for one instance.

#### 3.2.1. Format

```
<select type=QName match= QName action= QName>
  select body
</select>
```

#### 3.2.2. Position

This element is a top-level element of one query data, and may appear as a child of the <pmql> or <cascades> element.

#### 3.2.3. type Attribute

This attribute defines type of query and takes following values:

- entity: This type of query tries to search instances that has the entity name with match attribute. It is possible to put where elements under this select element.
- attribute: This type of query tries to search instances that are referred by the attribute that is defined by match attribute. If the attribute is entity reference, the search result will be zero or one instance. If the attribute is an aggregate of entity reference, the

- search result will be more than zero instances.
- **inverse:** This type of query tries to search instances that are defined by the inverse attribute that is defined by match attribute. The search result will be more than zero instances.
- **defined:** This type refers predefined PMQL data that is stored in IFC Model Server. The match attribute must be urn of PMQL's name. The action attribute is ignored.

### 3.2.4. match Attribute

This attribute defines matching name for the query. This value may be entity type, attribute name, and inverse name, with type attribute value of entity, attribute, inverse, respectively.

### 3.2.5. action Attribute

This attribute defines type of operation for the query. This action attribute takes following values:

- **get:** This will select the matched partial model data and return to client.
- **delete:** This will delete the matched partial model data.
- **update:** This will update the matched partial model data. The update element must be appear under the select element that defines update in action attribute.
- **nop:** no operation.

## 3.3 <where>

This element describes a conditional expression such as specifying object ID, entity type, and describing WHERE expression. The conditional operation is applied to the parent **<select>** element.

### 3.3.1. Format

```
<where>
  where body
</where>
```

### 3.3.2. Position

This element may appear as child of the **<select>** element.

### 3.3.3. op Attribute

This attribute defines a Boolean operator between two where elements. The default value is "AND". Following operators may appear in this attribute:

- **NOT:** Boolean NOT. This must be appeared only in the first where element.
- **AND:** Boolean AND.
- **OR:** Boolean OR.

## 3.4 <oidis>

This element defines a specific instance by the object ID. It is possible to define multiple object IDs by set of oids elements.

### 3.4.1. Format

```
<oidis value="045215AC-D287-B74F-8F26-E678DDA91A72"/>
```

### 3.4.2. Position

This element may appear as child of **<where>** element.

### 3.4.3. value Attribute

This attribute defines the object ID value.

### 3.4.4. Example

The following oidis select an instance that has the defined object ID.

```
<select type="entity" match="IfcWall" action="get">
  <where>
    <oidis value="045215AC-D287-B74F-8F26-E678DDA91A72"/>
  </where>
</select>
```

## 3.5 <typeis>

This element defines entity type of the select operation.

### 3.5.1. Format

```
<typeis value="IfcLocalPlacement"/>
```

### 3.5.2. Position

This element may appear as child of **<where>** element.

### 3.5.3. value Attribute

This attribute defines the entity type.

### 3.5.4. Example

The following typeis selects IfcLocalPlacement entity. This can apply to avoid to select IfcConstraintPlacement.

```
<select type="attribute" match="LocalPlacement" action="get">
  <where>
    <typeis value="IfcLocalPlacement"/>
  </where>
</select>
```

## 3.6 <expr>

This element defines an expression string of SQL. It is possible to contain multiple expressions into one expr element, because the expression string just is passed into database.

### 3.6.1. Format

```
<expr value="RelationshipType LIKE 'BuildingContainer'"/>
```

### 3.6.2. Position

This element may appear as child of **<where>** element.

### 3.6.3. value Attribute

This attribute defines the expression string.

### 3.6.4. Example

```
<where>
  <expr value="RelationshipType LIKE 'BuildingContainer'"/>
</where>
<where op="OR">
  <expr value="RelationshipType LIKE 'BuildingStoreyContainer'"/>
</where>
```

This is equal to:

```
<where>
```

```
<expr value="RelationshipType LIKE 'BuildingContainer' OR
RelationshipType LIKE 'BuildingStoreyContainer'"/>
</where>
```

### 3.7 <cascades>

This element contains children of select element repeatedly.

#### 3.7.1. Format

```
<cascades>
  cascades body
</cascades>
```

#### 3.7.2. Position

This element may appear as child of <select> element.

#### 3.7.3. Example

```
<select type="entity" match="IfcWall" action="get">
  <cascades>
    <select type="inverse" match="IsDefinedBy" action="get">
      <cascades>
        <select type="attribute" match="RelatingPropertyDefinition"
action="get"/>
      </cascades>
    </select>
  </cascades>
</select>
```

### 3.8 <update>

This element contains Enhanced BLIS-XML data to update the selected instance.

#### 3.8.1. Format

```
<update>
  Enhanced BLIS-XML body (EBLIS-XML)
</update>
```

#### 3.8.2. Position

This element may appear as child of <select> element that has update value in the action attribute.

#### 3.8.3. Example

```
<select type="entity" match="IfcWall" action="update">
  <where>
    <oidis value="045215AC-D287-B74F-8F26-E678DDA91A72"/>
  </where>
  <update>
    <IfcWall GlobalId="bHTEQ7&t:Zai:6;v|*@"
OwnerHistory="76071D0C-BDD4-3E47-844B-B5402144ECDA" Label="Wall#173"
UserDefinedType="" LocalPlacement="C5E3D3CE-E3FC-2E4D-B1A3-C8697DD5FEB7"
LayerInformation="56AE98A5-41B7-FD43-BBFF-3BEE182C0DE3"
calcWallArea="2.8733571093557599" calcWallVolume="0.68960569083149303">
      <Representations EntityRef="3823DC67-6316-5248-881F-141E38F60527" />
    </IfcWall>
  </update>
</select>
```

## 4. WHERE Element

In this section, we will see usage of **<where>** element and how make where rules to get partial model.

Basically, **<where>** element contains following child elements:

- **<oidis>**
- **<typeis>**
- **<expr>**

These child elements allows you to define the conditions that select specific objects. The **oidis** defines the matching object ID, the **typeis** defines the matching entity type, and the **expr** defined general conditional expressions based on SQL.

In one **<where>** element, the order of interpretation is, from highest to lowest, **oidis**, **typeis**, and **expr**. You can put **oidis** element after **typeis** or **expr**, but the **oids** element is taken before **typeis** and **expr**.

### 4.1 Boolean operation between where elements

A **<select>** element can contain multiple **<where>** elements. Each **<where>** element are joined by Boolean AND operator as default. The **op** attribute defines the Boolean operation. For example the operation between **<where>** elements are as follows:

- AND (default):
 

```
<where>
  where 1
</where>
<where>
  where 2
</where>
```

 is equal to, `where 1 AND where 2.`
- OR:
 

```
<where>
  where 1
</where>
<where op="OR">
  where 2
</where>
```

 is equal to, `where 1 OR where 2.`
- NOT:
 

```
<where op="NOT">
  where 1
</where>
```

 is equal to, `NOT where 1.`

### 4.2 Boolean operation between oidis elements

Each **<oidis>** element is joined by Boolean OR operator as default. For example the operation between **<oidis>** elements are as follows:

```
<where>
  <oidis value="OID1">
  <oidis value="OID2">
</where>
```

is equal to, `OID1 OR OID2.`

### 4.3 Boolean operation between typeis elements

Each **<typeis>** element is joined by Boolean OR operator as default. For example the operation between **<typeis>** elements are as follows:

```
<where>
```

```
<typeis value="ENTITY_TYPE1">  
<typeis value="ENTITY_TYPE2">  
</where>
```

is equal to, ENTITY\_TYPE1 OR ENTITY\_TYPE2.

## 4.4 Boolean operation between expr elements

Each **<expr>** element is joined by Boolean AND operator as default. For example the operation between **<expr>** elements are as follows:

```
<where>  
  <expr value="EXPRESSION1">  
  <expr value="EXPRESSION2">  
</where>
```

is equal to, EXPRESSION1 AND EXPRESSION2.

If you want to define OR operation, it may be as follows:

```
<where>  
  <expr value="EXPRESSION1 OR EXPRESSION2">  
</where>
```

or

```
<where>  
  <expr value="EXPRESSION1">  
</where>  
<where op="OR">  
  <expr value="EXPRESSION2">  
</where>
```

## 5. PMQL Examples

### 5.1 Select by Entity Type

The following PMQL selects every IfcSpace object and referred objects by their LocalPlacement and BoundedBy attributes. To apply this PMQL to the IFC model data, IfcSpace, IfcLocalPlacement and IfcSpaceBoundary objects would be selected. In this example, there is no where element.

```
<pmql>
<select type="entity" match="IfcSpace" action="get">
  <cascades>
    <select type="attribute" match="LocalPlacement" action="get"/>
    <select type="attribute" match="BoundedBy" action="get"/>
  </cascades>
</select>
</pmql>
```

### 5.2 Select with Where rule

The following PMQL selects IfcWall objects that have the Label attribute value is starting "Wall#", and calcWallVolume attribute value is less than 3.0.

```
<pmql>
<select type="entity" match="IfcWall" action="get">
  <where>
    <expr value="Label LIKE 'Wall#%'" />
    <expr value="calcWallVolume < 3.0" />
  </where>
</select>
</pmql>
```

### 5.3 Select with OID

The following PMQL selects an IfcWall object that has the specific object id that is defined by the oidis element.

```
<pmql>
<select type="entity" match="IfcWall" action="get">
  <where>
    <oidis value="045215AC-D287-B74F-8F26-E678DDA91A72" />
  </where>
</select>
</pmql>
```

### 5.4 Select with Inverse

The following PMQL select all IfcWall objects and related property set objects without extended property sets. The where element that is located under the select element that selects the entity reference of RelatingPropertyDefinition attribute, has a where element that limits the referenced entity must be IfcPropertySet object. It is possible to limit much more specific property set to additional where elements, for instance, by the name of property set.

```
<pmql>
<select type="entity" match="IfcWall" action="get">
  <cascades>
    <select type="inverse" match="IsDefinedBy" action="get">
      <cascades>
        <select type="attribute" match="RelatingPropertyDefinition"
action="get">
          <where>
```

```

        <typeis value="IfcPropertySet"/>
    </where>
    <cascaes>
        <select type="attribute" match="HasProperties"
action="get"/>
    </cascaes>
    </select>
</cascaes>
</select>
</cascaes>
</select>
</pmql>

```

## 5.5 Using Predefined PMQL

The IFC Model Server provides PMQL a functionality that stores PMQL data in the database. The client user or application can utilize such predefined PMQL by specifying the urn.

If the following PMQL is stored in the database,

```

<pmql name="urn:model-server:imsvr:ifc20_longform:ifcspace:basic">
<select type="entity" match="IfcSpace" action="get">
    <cascaes>
        <select type="attribute" match="LocalPlacement" action="get"/>
        <select type="attribute" match="BoundedBy" action="get"/>
    </cascaes>
</select>
</pmql>

```

you can call the predefined PMQL as follows:

```

<pmql>
    <select type="defined"
match="urn:model-server:imsvr:ifc20_longform:ifcspace:basic"/>
</pmql>

```