THE Open GROUP

FACE™ (Future Airborne Capability Environment)

Shared Data Model Governance Plan, Edition 3.1

February 2020

FACE™ (Future Airborne Capability Environment):
Shared Data Model Governance Plan, Edition 3.1

Document Number: X203US

Authored by The Open Group FACE Consortium.
Published by The Open Group, February 2020.

Comments relating to the material contained in this document may be submitted to:

The Open Group, 800 District Avenue, Suite 150, Burlington, MA 01803, United States

or by electronic mail to:

ogface-admin@opengroup.org

Contents

1. Introduction 4
   - 1.1 Scope 4
   - 1.2 FACE Technical Standard Edition-Specific Considerations 4

2. FACE SDM Editions, Versioning, and Releases 5
   - 2.1 Versions & Editions 5
     - 2.1.1 Editions 5
     - 2.1.2 Versions 6
   - 2.2 Release Cycle 6
   - 2.3 Backward Compatibility 6
   - 2.4 Distribution 6

3. FACE Shared Data Model Configuration Control Board 7
   - 3.1 Membership 7
   - 3.2 Roles and Responsibilities 8
     - 3.2.1 SDM CCB Chair 8
     - 3.2.2 FACE DIOG SDM Subcommittee Leads 8
     - 3.2.3 Members 8
   - 3.3 Voting 8

4. FACE SDM Managed Elements 10
   - 4.1 SDM Elements 10
   - 4.2 Deprecated Elements 10
   - 4.3 FACE Data Model Language 10

5. Change Request Process 11
6. Data Model Conformance 12
   - 6.1 Additional SDM Types 12
   - A.1 Basis Elements 13
   - A.2 Constraints for *face* Package 13
   - A.3 Constraints for *face::logical* Package 19
   - A.4 Constraints for *face::platform* Package 22
   - A.5 Constraints for *face::uop* Package 28
   - B.1 Basis Elements 31
   - B.2 Constraint Helper Methods 32
   - B.3 Constraints for *face* Package 34
   - B.4 Constraints for *face::conceptual* Package 35
   - B.5 Constraints for *face::logical* Package 41
   - B.6 Constraints for *face::platform* Package 49
   - C.1 Basis Elements 59
   - C.2 Query Rules 60
   - C.3 Template Rules 72
   - D.1 Basis Elements 100
   - D.2 Query Rules 101
   - D.3 Template Rules 101

1. Introduction

1.1 Scope

This Governance Plan defines the policies, processes, and mechanisms governing the Future Airborne Capability Environment (FACE) Shared Data Model (SDM) and establishes the SDM Configuration Control Board (SDM CCB). The SDM CCB derives its authority from the FACE Consortium Steering Committee through a majority vote to approve this plan. The scope of responsibility of the SDM CCB is to manage Change Requests (CRs) for the FACE SDM, document SDM updates associated with each edition of the FACE Technical Standard, and maintain configuration management for the various editions of the SDM. The plan also defines membership, roles, responsibilities, rules, and process flow for the SDM CCB.

1.2 FACE Technical Standard Edition-Specific Considerations

Particular considerations specific to each edition of the FACE Technical Standard are documented in an appendix (e.g., Appendix A corresponds to the FACE Technical Standard, Edition 2.0).[1]

[1] For all editions of the FACE Technical Standard, visit www.opengroup.org/face, select "Documents and Tools" and then the "FACE Publications" button.

2. FACE SDM Editions, Versioning, and Releases

The FACE Consortium will maintain two separate SDMs:

- The ITAR SDM contains International Traffic in Arms Regulations (ITAR) restricted information
- The Public SDM contains publicly releasable information and is a subset of the ITAR SDM, as depicted in Figure 1

The FACE Consortium will limit the differences between the ITAR SDM and Public SDM to information restricted by ITAR. The use of either the ITAR SDM or Public SDM is acceptable for achieving FACE conformance verification.

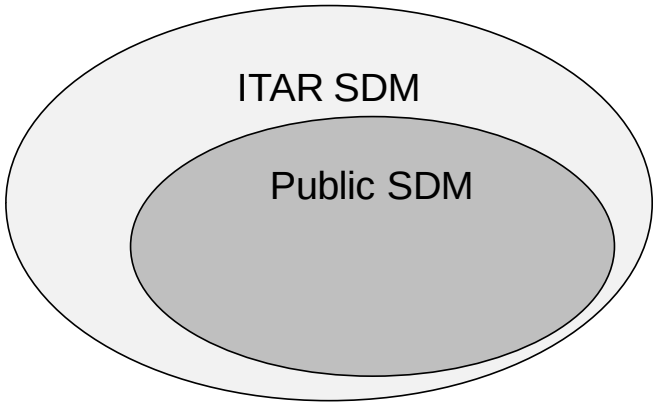

Figure 1: FACE SDMs

A version of the ITAR SDM is accessible to FACE Consortium members immediately upon being released. A Public SDM release requires Government approval by a Public Affairs Office.

Some data types may be restricted from public release due to their sensitive nature. A review of FACE Technical Standard data types suggests that small subsets of metamodel types are the most likely candidates to prevent public release. A list of specific data types for each edition of the FACE Technical Standard is presented in an appendix (e.g., Appendix A corresponds to FACE Technical Standard, Edition 2.0).

There may be extensions to the ITAR SDM that are not releasable due to security restrictions. Any such models are referred to as ITAR SDM + extensions.

2.1 Versions & Editions

Every release of an SDM is uniquely identified by an identifier (e.g., v2.1.35) indicating its edition and version.

2.1.1 Editions

An edition of the ITAR SDM and Public SDM will be created for each major edition of the FACE Technical Standard. The first two digits of an SDM's identifier indicate its edition. For example, Public SDM v2.1.x corresponds to Edition 2.1 of the FACE Technical Standard. Each edition of the SDM is

forwarded for FACE Consortium Steering Committee approval prior to publishing that edition of the SDM.

2.1.2 Versions

An SDM edition may be updated by the SDM CCB to resolve CRs submitted as part of the Change Request Process (see Section 5). Each update corresponds to a new version of the SDM. For example, Public SDM v2.1.5 is the fifth revision of the Public SDM, edition 2.1.

2.2 Release Cycle

The ITAR SDM release cycle is driven by CR frequency. Resolutions for multiple CRs may be included in a single release. An ITAR SDM may be approved for release as a Public SDM by the Government. A Public SDM release has the same edition and version as the ITAR SDM on which it is based.

2.3 Backward Compatibility

No changes, updates, or deletions should be made to an SDM that would require changes to a Unit of Portability Supplied Model (USM) or Domain-Specific Data Model (DSDM) developed against one version of an SDM when moving that USM or DSDM to a newer version of the SDM (within the same edition).

2.4 Distribution

Releases of the ITAR SDM will be made available to the FACE Consortium-only portion of The Open Group FACE website. Releases of the Public SDM will be made available on The Open Group FACE public website at https://www.opengroup.us/face/documents.php?action=show&dcat=&gdid=21020.

3. FACE Shared Data Model Configuration Control Board

3.1 Membership

The SDM CCB official voting membership consists of the FACE Technical Working Group (TWG) Chair, FACE TWG Vice-Chair, Domain Interoperability Working Group (DIOG) Chair, DIOG Vice-Chair, TWG Transport Subcommittee Lead, TWG Transport Subcommittee Co-Lead, DIOG SDM Subcommittee Lead, and DIOG SDM Subcommittee Co-Lead.

In addition to official voting members, the following additional stakeholders shall receive communication of all significant SDM CCB events and have an opportunity to review and comment on all significant SDM CCB decisions and artifacts.

In the event one member of the SDM CCB holds more than one position stated above, the SDM CCB will appoint a substitute member from the list of stakeholders.

- Stakeholders from the FACE TWG:
  - General Enhancements Subcommittee Lead or delegate
  - Conformance Verification Matrix Subcommittee Lead or delegate
  - Graphics Subcommittee Lead or delegate
  - Security Subcommittee Lead or delegate
  - Software Safety Subcommittee Lead or delegate
- Stakeholders from the FACE Business Working Group:
  - Strategy Subcommittee Lead or delegate
  - Business Operations Subcommittee Lead or delegate
  - Outreach Subcommittee Lead or delegate
- Stakeholders from the FACE Domain Interoperability Working Group:
  - Language Subcommittee Lead or delegate
  - Guidance Subcommittee Lead or delegate
- Stakeholders from the Standards Alignment Advisory Group(s):
  - The appointed Advisory Group members from the FACE Consortium
  - The appointed Advisory Group members from the collaborating Standards Body(ies)

3.2 Roles and Responsibilities

3.2.1 SDM CCB Chair

The SDM CCB Chair shall be the FACE DIOG Chair. The SDM CCB Chair will be the primary facilitator of the SDM CCB performing the following duties:

- Coordinate with DIOG SDM Subcommittee Leads to retrieve CRs and work with submitter(s) to clarify the CR
- Coordinate with DIOG SDM Subcommittee Leads to determine and set the length of the review time
- Coordinate with DIOG SDM Subcommittee Leads to initiate the review process described in Section 5
- Vote as described in Section 3.3

3.2.2 FACE DIOG SDM Subcommittee Leads

The FACE DIOG SDM Subcommittee Leads shall ensure the following duties are performed by the DIOG SDM Subcommittee:

- Implement approved CR as quickly as possible
- Work with the SDM CCB to verify implementation
- Maintain configuration management of the FACE SDM
- Notify FACE TWG membership and FACE leadership of impending changes

3.2.3 Members

Members of the SDM CCB have the following duties:

- Vote on matters as they are brought forward
- Failure to vote within the time specified results in an Abstain vote being logged on the CR

3.3 Voting

Members of the SDM CCB vote using one of the following four choices:

- Approve
- Accept (with Modification)
- Reject (with Rationale)
- Abstain

A CR must gain 75% of the SDM CCB voting members' approval, defined as Approve and Accept (with Modification) votes, and have more Approve votes than Reject and Accept with Modification votes combined for the CR to pass. If the request is not approved and there are Accept with Modification vote(s), a requestor may initiate a new CR vote based on the Modification(s) proposed. If there are multiple Accept with Modification votes they may be consolidated for subsequent votes or each submitted as a separate CR at the discretion of the requestor(s). A member of the SDM CCB may abstain from

voting when their organization has an interest, either as a submitter or potential competitor, in the outcome of the CR that is being considered by the SDM CCB. This would potentially alleviate any perception that their organization has an advantage due to an employee being a member of the SDM CCB. It allows the SDM CCB to maintain impartiality during the CR process. Preserving the impartiality of the SDM CCB is an important consideration as the CCB impacts the SDM which is a key component of the FACE Reference Architecture.

The requester may withdraw a CR at any time prior to approval or rejection.

The option to Accept with Modification must include details of the Modification proposed. The option to reject must include a rationale for rejection. Failure to provide adequate Modification or Rationale, as determined by the SDM CCB Chair, results in an Abstain vote.

4. FACE SDM Managed Elements

4.1 SDM Elements

A Basis Element (defined in the appendices) is a type of Data Model element that must reside in the SDM. Basis Elements are managed by the SDM CCB. Non-Basis Elements are not required to reside in the SDM, but can be added if there is potential for reuse. Once added to the SDM, non-Basis Elements are managed by the SDM CCB.

Note: "Basis Element" is not to be confused with similarly named metatypes in the FACE Data Model Language (i.e., Basis Entity in Edition 3.0 of the FACE Technical Standard).

4.2 Deprecated Elements

Any data model element may be deprecated by the SDM CCB. Deprecated elements in one edition of the FACE SDM are intended to be removed from the next edition. The use of deprecated elements is discouraged.

The FACE Data Model Language in the FACE Technical Standard, Edition 2.0 includes a flag on model elements to indicate deprecation. The language in the FACE Technical Standard, Edition 2.1, Edition 3.0, and Edition 3.1 does not have this feature; instead, an element's description is used to indicate the element is deprecated and to provide a recommended replacement (where possible). Such elements may also be moved to Conceptual, Logical, or Platform Data Models with a name prefixed "Deprecated" – any contained element is considered deprecated.

4.3 FACE Data Model Language

Changes to the FACE Data Model Language are not the responsibility of the SDM CCB but are managed by the FACE Consortium. They are therefore the responsibility of the FACE Domain Interoperability Working Group, the FACE TWG, and the FACE Consortium Steering Committee.

5. Change Request Process

Any person or organization may submit a CR to recommend FACE SDM additions or corrections. All CRs shall be submitted through the FACE Consortium PR/CR System and will follow the approved PR/CR process as defined by the FACE Problem Report and Change Request Process.[2] The CR must include an attachment containing a well-formed FACE Data Model that does not contain ITAR data, and must identify and include a detailed description of the proposed changes.

If the CR reaches the SDM CCB to be investigated, CRs will be evaluated and voted upon by CCB members and, if approved, will be implemented by the FACE DIOG SDM Subcommittee Leads. Multiple CR implementations may be aggregated into a single release of an SDM, depending on CR submission frequency and resource availability.

The CCB will not approve modifications to the SDM outside those required to implement a CR.

[2] Future Airborne Capability Environment (FACE™) Problem Report (PR) and Change Request (CR) Process (G166), published by The Open Group, September 2016; refer to: www.opengroup.org/library/g166.

6. Data Model Conformance

A USM or DSDM is verified for conformance to the FACE Data Architecture Requirements as defined in the corresponding edition of the FACE Technical Standard (i.e., Edition 2.0 §3.6.2; Edition 2.1 §3.6.2; Edition 3.0 §3.9.4; Edition 3.1 §4.9.4). Additionally, a USM or DSDM is verified to be consistent with the appropriate SDM as part of the conformance verification process. Consistency is checked by comparing the USM or DSDM elements to the appropriate SDM elements.

Note: USMs are only verified for conformance along with the corresponding PCS, PSSS, or TSS UoC.

The Software Supplier must state which SDM is to be used for conformance verification and which edition/version. The SDM options are:

- ITAR SDM
- Public SDM
- ITAR SDM + extensions

For purposes of comparison, the following are definitions of additions and updates:

- Additions are elements in which the xmi:id of the element in the USM or DSDM is not present in the SDM
- Updates are elements in which the xmi:id of the element in the USM or DSDM is present in the SDM and the element in the USM or DSDM is not identical to the element in the SDM

If a USM or DSDM contains Basis Elements (defined in the relevant appendix) that are additions or updates, the USM or DSDM will fail the conformance verification process. There are some Basis Elements that may be added and not cause conformance failure if the Software Supplier can provide evidence as to why they are not releasable and if "ITAR SDM + extensions" is being used.

The FACE Conformance Test Suite verifies additions or updates to Basis Elements. Any additions to these types (in a USM or DSDM) will be identified by the FACE Conformance Test Suite(s) and will show up as information in the resulting test report. They will trigger the Verification Authority to review the elements.

6.1 Additional SDM Types

In addition to the Basis Elements of the FACE SDM, other types in a USM or DSDM will be checked against the SDM in the conformance process. The USM or DSDM may have additional information but may not update existing elements in the SDM. Each element in a USM or DSDM contains a unique identifier (xmi:id) that will be used during the conformance verification process to determine whether an element in the SDM was renamed or modified.

A FACE Technical Standard, Edition 2.0 Data Types

A review of FACE Technical Standard, Edition 2.0 data types suggests that the following data types are the most likely candidates to prevent public release:

- face.logical.FrameOfReference
- face.logical.Enumeration
- face.logical.SimpleMeasurement
- face.logical.CompositeMeasurement

The following data type is also included as it is an abstract base type for face.logical.SimpleMeasurement and face.logical.CompositeMeasurement and could therefore contain sensitive information:

- face.logical.Measurement

A.1 Basis Elements

The following elements are the Basis Elements for FACE Edition 2.0:

- face.conceptual.Observable
- face.conceptual.InformationElement
- face.logical.Unit
- face.logical.Conversion (unit-to-unit conversions only)
- face.logical.Measurement (Abstract)
- face.logical.FrameOfReference
- face.logical.Enumeration
- face.logical.SimpleMeasurement
- face.logical.CompositeMeasurement

A.2 Constraints for *face* Package

```
package face

  context Element

    /*
     * 'tokenizeString' will return a sequence of strings that have been
     * split from the input string by the delimiter.
     * Assumes single character delimiter.
     */
    static def: tokenizeString(str : String, delimiter : String) : Sequence(String) =
      let i : Integer = str.indexOf(delimiter) in
      if i > 1
      then
          let token : String = str.substring(1, i-1) in
          if i = str.size()
          then
```

```
            Sequence{token}
          else
            let remainder : String = str.substring(i+1, str.size()) in
            let remainderTokens : Sequence(String) = tokenizeString(remainder,
                delimiter) in
            remainderTokens->prepend(token)
          endif
        else
          if i = str.size()
          then
            Sequence{}
          else
            if i = 1
            then
              let remainder : String = str.substring(i+1, str.size()) in
              self.tokenizeString(remainder, delimiter)
            else -- i <= 0
              Sequence{str}
            endif
          endif
        endif

endpackage

package face::conceptual

  context Element
    /*
     * Every face::conceptual::Element in a FACE data model shall
     * have a unique name.
     */
    inv hasUniqueName:
      let conceptualElements: Set(face::conceptual::Element) =
        face::conceptual::Element.allInstances() in
      let otherConceptualElements: Set(face::conceptual::Element) =
        conceptualElements->excluding(self) in
      not otherConceptualElements->collect(name)->exists(e | e = self.name)

  context Entity

    /*
     * A helper method that returns the Identity of an Entity
     */
  def: getEntityIdentity() : Bag(OclAny) =
    let compositions : Bag(face::conceptual::Characteristic) =
      self->collect(composition)
    in

    let characteristics : Bag(face::conceptual::Characteristic) =
      if self.oclIsTypeOf(face::conceptual::Association)
      then
        self.oclAsType(face::conceptual::Association)->
          collect(associationEnd.oclAsType(face::conceptual::Characteristic))->
          union(compositions)
      else
        compositions
      endif
    in

      characteristics->collect(c | c.getIdentityContribution())

    /*
     * The full composition hierarchy of each conceptual Entity shall be
```

```
    * unique. Uniqueness is determined by number, multiplicity, and
    * type of its composed ComposableElement.
    */
inv entityIsUnique:
  let ceIdentity: Bag(OclAny) =
    self.getEntityIdentity() in
    face::conceptual::Entity.allInstances()->excluding(self)
      ->forAll(ce | ce.getEntityIdentity() <> ceIdentity
      )

  /*
   * Every face::conceptual::Entity in a FACE data model shall contain
   * a face::conceptual::InformationElement named 'UniqueIDType'.
   */
  inv hasUniqueID:
    self->collect(composition)->collect(type)
      ->exists(a | a.oclIsTypeOf(Observable)
        and a.name = 'UniqueIdentifier'
      )

  /*
   * Every face::conceptual::Composition within the scope
   * of an Entity must have a unique name.
   */
  inv allCompositionsHaveUniqueRolename:
    self->collect(composition)->isUnique(rolename)

  /*
   * A conceptual Entity must be composed from conceptual
   * BasisElements or other conceptual Entities which
   * are composed of conceptual BasisElements.
   */
  inv entityConstructed:
    let ceClosure = self->closure(ce |
      let types = ce->collect(composition)->collect(type)->asSet()
        in
      let entityTypes = types
        ->select(t | t.oclIsTypeOf(face::conceptual::Entity)) in
      entityTypes->collect(t | t.oclAsType(face::conceptual::Entity))
    ) in
    not ceClosure->includes(self)

context Association
  /*
   * Every face::conceptual::Characteristic within the scope
   * of an Association must have a unique name.
   */
  inv allCharacteristicsHaveUniqueRolename:
    let compositions =
      self->collect(composition)->
            collect(c|c.oclAsType(face::conceptual::Characteristic)) in
    let associatedEntities =
      self->collect(associationEnd)->
            collect(c|c.oclAsType(face::conceptual::Characteristic)) in
  let characteristics: Set(face::conceptual::Characteristic) =
    compositions->union(associatedEntities) in
  characteristics->isUnique(rolename)

context View

  /*
   * A helper method that returns the Identity of an Entity.
   */
```

```
def: getViewIdentity() : Bag(OclAny) =
  self->collect(characteristic)->collect(c | c.getIdentityContribution())

  /*
   * The each View shall be unique. Uniqueness is determined by
   * number, multiplicity, type, and context of its ProjectedCharacteristics.
   */
  inv viewIsUnique:
  let cvIdentity: Bag(OclAny) =
    self.getViewIdentity() in
    face::conceptual::View.allInstances()->excluding(self)
      ->forAll(cv | cv.getViewIdentity() <> cvIdentity
      )

  /*
   * The each View shall be unique. Uniqueness is determined by
   * number, multiplicity, type, and context of its ProjectedCharacteristics.
   */
  inv viewIsUnique2:
    true

context Characteristic

  /*
   * A helper method that returns the contribution that
   * a Characteristic makes to an Entity's identity.
   */
  def: getIdentityContribution() : Sequence(OclAny) =
  let type : face::conceptual::ComposableElement =
    if self.oclIsTypeOf(face::conceptual::Composition)
    then
      self.oclAsType(face::conceptual::Composition).type
    else
      self.oclAsType(face::conceptual::AssociatedEntity).type
    endif
  in

  Sequence{type, self.upperBound, self.lowerBound}

context Composition
  /*
   * For conceptual, logical, and platform Compositions the lowerBound
   * shall be less than or equal to upperBound.
   */
  inv lowerBound_LTE_UpperBound:
    self.upperBound <> -1 implies self.lowerBound <= self.upperBound
  /*
   * For conceptual, logical, and platform Compositions, upperBound shall
   * be == -1 or >= 1.
   */
  inv upperBoundValid:
    self.upperBound = -1 or self.upperBound >= 1
  /*
   * For conceptual, logical, and platform Compositions, lowerBound shall
   * be >= zero.
   */
  inv lowerBoundValid:
    self.lowerBound  >= 0

context CharacteristicProjection

  /*
   * A helper method that returns the contribution that
```

```
    * a Characteristic makes to an Entity's identity.
    */
  def: getIdentityContribution() : Sequence(OclAny) =
  Sequence{self.projectedCharacteristic, self.path}

  /*
   * Helper method to remove first node from a sequence of strings
   */
def: subSequenceStrings(s : Sequence(String)) : Sequence(String) =

  if s->size() = 1 then s->select(false) else s->subSequence(2, s->size()) endif

  /*
   * Helper method to determine if an association path resolution and subsequent
   * path selectors are valid from a relative location.
   */
def: associationPathResolutionValid(entity : face::conceptual::Entity, pathTokens :
  Sequence(String)) : Boolean =

  let currPathTokenSplit : Sequence(String) =
          Element::tokenizeString(pathTokens->first().replaceAll(']', ''),'[') in
  let currPathTokenRolename : String = currPathTokenSplit->first() in
  let currPathTokenAssociation : String = currPathTokenSplit->last() in
  let allAssociations : Collection(face::conceptual::Association) =
    face::conceptual::Association.allInstances() in

  -- there should be exactly one Association with expected name
  if (allAssociations->one(a | a.name = currPathTokenAssociation))
  then (
    let association : face::conceptual::Association =
      allAssociations->any(a | a.name = currPathTokenAssociation) in

    -- the Association should have exactly one AssociatedEntity
    -- with expected rolename
    if association.associationEnd->one(c | c.rolename = currPathTokenRolename)
    then (

      -- get the one AssociatedEntity with expected rolename
      let comp : face::conceptual::AssociatedEntity =
        association.associationEnd->any(c | c.rolename = currPathTokenRolename) in

      -- composition is valid if it exists, has a type, and the type is
      -- the current entity
      let compTypeValid : Boolean =
        comp <> null and comp.type <> null and comp.type = entity in

      -- path is valid it current node is valid and all subsequent nodes are valid
      compTypeValid and pathValid(association, subSequenceStrings(pathTokens))
    )
    else
      -- path is invalid if there is not exactly one AssociatedEntity
      -- with expected rolename
      false
    endif
  )
  else
    -- path is invalid if there is not exactly one Association
    -- with expected name
    false
  endif

  /*
   * Helper method to determine if a composition path resolution and subsequent
```

```
     * path selectors are valid from a relative location.
     */
  def: compositionPathResolutionValid(entity : face::conceptual::Entity, pathTokens :
    Sequence(String)) : Boolean =

    -- there should be exactly one Characteristic with expected rolename
    -- check if that Characteristic is a Composition
    if entity.composition->one(rolename = pathTokens->first())
    then (
      -- get composition with expected rolename
      let comp : face::conceptual::Composition =
        entity.composition->any(rolename = pathTokens->first()) in

      -- get associatedEntity with expected rolename
      let associatedEntity : face::conceptual::AssociatedEntity =
        entity.composition->any(rolename = pathTokens->first()) in

      -- path is valid it current node is valid and all subsequent nodes are valid
      self.pathValid(comp.type, subSequenceStrings(pathTokens))
    )
    else
      -- Check if the Characteristic with expected rolename is an AssociatedEntity
      if (entity.oclIsTypeOf(face::conceptual::Association) and
        entity.oclAsType(face::conceptual::Association).
          associationEnd->one(rolename = pathTokens->first())
      )
      then (
        -- get associatedEntity with expected rolename
        let associatedEntity : face::conceptual::AssociatedEntity =
          entity.oclAsType(face::conceptual::Association).
            associationEnd->any(rolename = pathTokens->first()) in

        -- path is valid it current node is valid and all subsequent nodes are valid
        self.pathValid(associatedEntity.type, subSequenceStrings(pathTokens))
)
      else
        -- path is invalid if there is not exactly
        -- one Characteristic with expected rolename
        false
      endif
    endif

    /*
     * Helper method to determine if a path relative to a ComposableElement is valid.
     */
  def: pathValid(ce : face::conceptual::ComposableElement, pathTokens :
    Sequence(String)) : Boolean =

    if (ce = null)
    then
      -- if ce is null then the path is invalid
      false
    else
      if pathTokens->size() = 0
      then
        -- if there are no more path tokens then the projection is valid
        true

      else (
        -- more path tokens indicate this must be an entity
        if ce.oclIsKindOf(face::conceptual::Entity)
        then (
          let entity : face::conceptual::Entity =
```

```
            ce.oclAsType(face::conceptual::Entity) in

          -- association resolutions have a square brackets
          if pathTokens->first().indexOf('[') > 0
          then
            associationPathResolutionValid(entity, pathTokens)
          else
            compositionPathResolutionValid(entity, pathTokens)
          endif
        )
        else (
          -- more path tokens indicate this must be an entity,
          -- if not then the path is invalid
          false
        )
        endif
      )
      endif
    endif

    /*
     * CharacteristicProjection must have a valid path.
     */
  inv characteristicProjectionPathValid:
    let ce : face::conceptual::ComposableElement =
      self.projectedCharacteristic.oclAsType(face::conceptual::ComposableElement) in
    let tokens : Sequence(String) =
      Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in
    self.pathValid(ce, tokens)

    /*
     * CharacteristicProjection must have a valid path format.
     */
  inv characteristicProjectionValidFormat:

    self.path <> null implies

    --remove all entity projection substrings (e.g., ".description")
    let pathTmp1 = self.path.replaceAll('\\.[_a-zA-Z0-9]+', '') in

    --remove all entity projection substrings (e.g., "->description[A1]")
    let pathTmp2 = pathTmp1.replaceAll('->[_a-zA-Z0-9]+\\[[_a-zA-Z0-9]+\\]', '') in

    pathTmp2.size() = 0

  context InformationElement
    /*
     * Information Elements will be deprecated in future versions of FACE.
     */
    inv informationElementDepricated:
      false

endpackage
```

A.3 Constraints for *face::logical* Package

```
package face::logical

  context Element
    /*
     * Every face::logical::Element except constraints in a FACE data model shall
     * have a unique name.
```

```
   */
  inv hasUniqueName:
    if self.oclIsKindOf(face::logical::Constraint)
    then
        -- do not check name on Constraint specification
        true
  else
      -- get all instances of face::logical::Element
      let logicalElements: Set(face::logical::Element) =
        face::logical::Element.allInstances() in
      let otherLogicalElements: Set(face::logical::Element) =
        logicalElements->excluding(self) in
      not otherLogicalElements->collect(name)->exists(e | e = self.name)
  endif

context Entity
  /*
   * Every face::logical::Composition within the scope
   * of an Entity must have a unique name.
   */
  inv allCompositionsHaveUniqueRolename:
    self->collect(composition)->isUnique(rolename)

context Association
  /*
   * Every face::logical::Characteristic within the scope
   * of an Association must have a unique name.
   */
  inv allCharacteristicsHaveUniqueRolename:
    let compositions =
      self->collect(composition)->
            collect(c|c.oclAsType(face::logical::Characteristic)) in
    let associatedEntities =
      self->collect(associatedEntity)->
            collect(c|c.oclAsType(face::logical::Characteristic)) in
  let characteristics: Set(face::logical::Characteristic) =
    compositions->union(associatedEntities) in
  characteristics->isUnique(rolename)

context Composition
/*
 * Ensure that when an element realizes another element, the
 * upper and lower bounds of the realized entity match those
 * of the realizing entity.
 */
inv logicalLowerBoundEqualsConceptual:
  let realizedComposition: face::conceptual::Composition = self.realizes in
  self.lowerBound = realizedComposition.lowerBound

inv logicalUpperBoundEqualsConceptual:
  let realizedComposition: face::conceptual::Composition = self.realizes in
  self.upperBound = realizedComposition.upperBound

/* A logical entity composition hierarchy must be consistent
 * with the composition hierarchy of the conceptual entity
 * that it realizes. The logical measurements must correspond
 * with the conceptual observables.
 */
inv logicalEntityConsistentWithConceptual:
  let realizedComposition: face::conceptual::Composition = self.realizes in
  let type: face::logical::ComposableElement = self.type in
  if type = null then
    false
```

```
    else
    if type.oclIsKindOf(face::logical::Entity) then
      let entityType: face::logical::Entity = type.oclAsType(face::logical::Entity) in
      let conceptualType: face::conceptual::Entity = entityType.realizes in
      conceptualType = realizedComposition.type
    else
    if type.oclIsKindOf(face::logical::Measurement) then
      let measurementType: face::logical::Measurement =
          type.oclAsType(face::logical::Measurement) in
      let conceptualType: face::conceptual::Observable = measurementType.realizes in
      conceptualType = realizedComposition.type
    else
    if type.oclIsKindOf(face::logical::InformationElement) then
      let ieType: face::logical::InformationElement =
          type.oclAsType(face::logical::InformationElement) in
      let conceptualType: face::conceptual::InformationElement = ieType.realizes in
      conceptualType = realizedComposition.type
    else
      false
    endif
    endif
    endif
    endif

  context CharacteristicProjection
    /*
     * CharacteristicProjection must have a valid path format.
     */
  inv characteristicProjectionValidFormat:

    self.path <> null implies

    --remove all entity projection substrings (e.g., ".description")
    let pathTmp1 = self.path.replaceAll('\\.[_a-zA-Z0-9]+', '') in

    --remove all entity projection substrings (e.g., "->description[A1]")
    let pathTmp2 = pathTmp1.replaceAll('->[_a-zA-Z0-9]+\\[[_a-zA-Z0-9]+\\]', '') in

    pathTmp2.size() = 0

  context Conversion
    /*
     * A face::logical::Conversion shall associate two
     * face::logical::ConvertableElement that are of the same metaclass
     * (e.g., only unit to unit or FrameOfReference to
     * FrameOfReference conversions are allowed).
     */
    inv unitConvertsToUnit:
      self.source.oclIsTypeOf(Unit) implies
        self.destination.oclIsTypeOf(Unit)
    inv frameOfReferenceConvertsToFrameOfReference:
      self.source.oclIsTypeOf(FrameOfReference) implies
        self.destination.oclIsTypeOf(FrameOfReference)

  context InformationElement
    /*
     * Information Elements will be deprecated in future versions of FACE.
     */
    inv informationElementDepricated:
      false

endpackage
```

A.4 Constraints for *face::platform* Package

```
package face::platform

  context Element
    /*
     * Every face::platform::Element in a FACE data model shall
     * have a unique name.
     */
    inv hasUniqueName:
      let platformElements: Set(face::platform::Element) =
        face::platform::Element.allInstances() in
      let otherPlatformElements: Set(face::platform::Element) =
        platformElements->excluding(self) in
      not otherPlatformElements->collect(name)->exists(e | e = self.name)

  context Composition
  /*
   * Ensure that when an element realizes another element, the
   * upper and lower bounds of the realized entity match those
   * of the realizing entity.
   */
  inv platformLowerBoundEqualsLogical:
    let realizedLogicalComposition: face::logical::Composition = self.realizes in
    self.lowerBound = realizedLogicalComposition.lowerBound

  inv platformUpperBoundEqualsLogical:
    let realizedLogicalComposition: face::logical::Composition = self.realizes in
    self.upperBound = realizedLogicalComposition.upperBound

  /* A platform entity composition hierarchy must be consistent
   * with the composition hierarchy of the logical entity
   * that it realizes. The platform value types must correspond
   * with the logical measurements and information elements.
   */
  inv platformEntityConsistentWithLogical:
    let realizedComposition: face::logical::Composition = self.realizes in
    let type: face::platform::ComposableElement = self.type in
    if type = null then
      false
    else
    if type.oclIsKindOf(face::platform::Entity) then
      let entityType: face::platform::Entity =
          type.oclAsType(face::platform::Entity) in
      let logicalType: face::logical::Entity = entityType.realizes in
      logicalType = realizedComposition.type
    else
    if type.oclIsKindOf(face::platform::IDLPrimitive) then
      let idlType: face::platform::IDLPrimitive =
          type.oclAsType(face::platform::IDLPrimitive) in
      let logicalType: face::logical::ValueElement = idlType.realizes in
      logicalType = realizedComposition.type
    else
    if type.oclIsKindOf(face::platform::IDLStruct) then
      let idlType: face::platform::IDLStruct =
          type.oclAsType(face::platform::IDLStruct) in
      let logicalType: face::logical::ValueElement = idlType.realizes in
      logicalType = realizedComposition.type
    else
      false
    endif
    endif
    endif
```

```
  endif

/*
 * Ensure that composition rolename does not conflict with
 * a reserved word in IDL or FACE supported programming
 * language.
 */

inv compositionNameNotReservedWord:
    let name: String = self.rolename.toLower() in
  name <> 'abstract' and
  name <> 'any' and
  name <> 'attribute' and
  name <> 'boolean' and
  name <> 'case' and
  name <> 'char' and
  name <> 'component' and
  name <> 'const' and
  name <> 'consumes' and
  name <> 'context' and
  name <> 'custom' and
  name <> 'default' and
  name <> 'double' and
  name <> 'emits' and
  name <> 'enum' and
  name <> 'eventtype' and
  name <> 'exception' and
  name <> 'factory' and
  name <> 'false' and
  name <> 'finder' and
  name <> 'fixed' and
  name <> 'float' and
  name <> 'getraises' and
  name <> 'home' and
  name <> 'import' and
  name <> 'in' and
  name <> 'inout' and
  name <> 'interface' and
  name <> 'local' and
  name <> 'long' and
  name <> 'manages' and
  name <> 'module' and
  name <> 'multiple' and
  name <> 'native' and
  name <> 'object' and
  name <> 'octet' and
  name <> 'oneway' and
  name <> 'out' and
  name <> 'primarykey' and
  name <> 'private' and
  name <> 'provides' and
  name <> 'public' and
  name <> 'publishes' and
  name <> 'raises' and
  name <> 'readonly' and
  name <> 'sequence' and
  name <> 'setraises' and
  name <> 'short' and
  name <> 'string' and
  name <> 'struct' and
  name <> 'supports' and
  name <> 'switch' and
  name <> 'true' and
```

```
    name <> 'truncatable' and
    name <> 'typedef' and
    name <> 'typeid' and
    name <> 'typeprefix' and
    name <> 'union' and
    name <> 'unsigned' and
    name <> 'uses' and
    name <> 'valuebase' and
    name <> 'valuetype' and
    name <> 'void' and
    name <> 'wchar' and
    name <> 'wstring'

context Entity

    /*
     * Every face::platform::Composition within the scope
     * of an Entity must have a unique name.
     */
    inv allCompositionsHaveUniqueRolename:
      self->collect(composition)->isUnique(rolename)

/*
 * Ensure that entity name does not conflict with
 * a reserved word in IDL or FACE supported programming
 * language.
 */

    inv entityNameNotReservedWord:
      let name: String = self.name.toLower() in
    name <> 'abstract' and
    name <> 'any' and
    name <> 'attribute' and
    name <> 'boolean' and
    name <> 'case' and
    name <> 'char' and
    name <> 'component' and
    name <> 'const' and
    name <> 'consumes' and
    name <> 'context' and
    name <> 'custom' and
    name <> 'default' and
    name <> 'double' and
    name <> 'emits' and
    name <> 'enum' and
    name <> 'eventtype' and
    name <> 'exception' and
    name <> 'factory' and
    name <> 'false' and
    name <> 'finder' and
    name <> 'fixed' and
    name <> 'float' and
    name <> 'getraises' and
    name <> 'home' and
    name <> 'import' and
    name <> 'in' and
    name <> 'inout' and
    name <> 'interface' and
    name <> 'local' and
    name <> 'long' and
    name <> 'manages' and
    name <> 'module' and
    name <> 'multiple' and
```

```
  name <> 'native' and
  name <> 'object' and
  name <> 'octet' and
  name <> 'oneway' and
  name <> 'out' and
  name <> 'primarykey' and
  name <> 'private' and
  name <> 'provides' and
  name <> 'public' and
  name <> 'publishes' and
  name <> 'raises' and
  name <> 'readonly' and
  name <> 'sequence' and
  name <> 'setraises' and
  name <> 'short' and
  name <> 'string' and
  name <> 'struct' and
  name <> 'supports' and
  name <> 'switch' and
  name <> 'true' and
  name <> 'truncatable' and
  name <> 'typedef' and
  name <> 'typeid' and
  name <> 'typeprefix' and
  name <> 'union' and
  name <> 'unsigned' and
  name <> 'uses' and
  name <> 'valuebase' and
  name <> 'valuetype' and
  name <> 'void' and
  name <> 'wchar' and
  name <> 'wstring'

context Association
  /*
   * Every face::logical::Characteristic within the scope
   * of an Association must have a unique name.
   */
  inv allCharacteristicsHaveUniqueRolename:
    let compositions =
      self->collect(composition)->
            collect(c|c.oclAsType(face::platform::Characteristic)) in
    let associatedEntities =
      self->collect(associatedEntity)->
            collect(c|c.oclAsType(face::platform::Characteristic)) in
  let characteristics: Set(face::platform::Characteristic) =
    compositions->union(associatedEntities) in
  characteristics->isUnique(rolename)

context View

/*
 * Ensure that view name does not conflict with
 * a reserved word in IDL or FACE supported programming
 * language.
 */

  inv viewNameNotReservedWord:
    let name: String = self.name.toLower() in
  name <> 'abstract' and
  name <> 'any' and
  name <> 'attribute' and
  name <> 'boolean' and
```

```
    name <> 'case' and
    name <> 'char' and
    name <> 'component' and
    name <> 'const' and
    name <> 'consumes' and
    name <> 'context' and
    name <> 'custom' and
    name <> 'default' and
    name <> 'double' and
    name <> 'emits' and
    name <> 'enum' and
    name <> 'eventtype' and
    name <> 'exception' and
    name <> 'factory' and
    name <> 'false' and
    name <> 'finder' and
    name <> 'fixed' and
    name <> 'float' and
    name <> 'getraises' and
    name <> 'home' and
    name <> 'import' and
    name <> 'in' and
    name <> 'inout' and
    name <> 'interface' and
    name <> 'local' and
    name <> 'long' and
    name <> 'manages' and
    name <> 'module' and
    name <> 'multiple' and
    name <> 'native' and
    name <> 'object' and
    name <> 'octet' and
    name <> 'oneway' and
    name <> 'out' and
    name <> 'primarykey' and
    name <> 'private' and
    name <> 'provides' and
    name <> 'public' and
    name <> 'publishes' and
    name <> 'raises' and
    name <> 'readonly' and
    name <> 'sequence' and
    name <> 'setraises' and
    name <> 'short' and
    name <> 'string' and
    name <> 'struct' and
    name <> 'supports' and
    name <> 'switch' and
    name <> 'true' and
    name <> 'truncatable' and
    name <> 'typedef' and
    name <> 'typeid' and
    name <> 'typeprefix' and
    name <> 'union' and
    name <> 'unsigned' and
    name <> 'uses' and
    name <> 'valuebase' and
    name <> 'valuetype' and
    name <> 'void' and
    name <> 'wchar' and
    name <> 'wstring'

  context CharacteristicProjection
```

```
/*
 * CharacteristicProjection must have a valid path format.
 */
inv characteristicProjectionValidFormat:

  self.path <> null implies

  --remove all entity projection substrings (e.g., ".description")
  let pathTmp1 = self.path.replaceAll('\\.[_a-zA-Z0-9]+', '') in

  --remove all entity projection substrings (e.g., "->description[A1]")
  let pathTmp2 = pathTmp1.replaceAll('->[_a-zA-Z0-9]+\\[[_a-zA-Z0-9]+\\]', '') in

  pathTmp2.size() = 0

  /*
   * Ensure that characteristic projection rolename does
   * not conflict with a reserved word in IDL or FACE
   * supported programming language.
   */

  inv characteristicProjectionNameNotReservedWord:
    let name: String = self.rolename.toLower() in
  name <> 'abstract' and
  name <> 'any' and
  name <> 'attribute' and
  name <> 'boolean' and
  name <> 'case' and
  name <> 'char' and
  name <> 'component' and
  name <> 'const' and
  name <> 'consumes' and
  name <> 'context' and
  name <> 'custom' and
  name <> 'default' and
  name <> 'double' and
  name <> 'emits' and
  name <> 'enum' and
  name <> 'eventtype' and
  name <> 'exception' and
  name <> 'factory' and
  name <> 'false' and
  name <> 'finder' and
  name <> 'fixed' and
  name <> 'float' and
  name <> 'getraises' and
  name <> 'home' and
  name <> 'import' and
  name <> 'in' and
  name <> 'inout' and
  name <> 'interface' and
  name <> 'local' and
  name <> 'long' and
  name <> 'manages' and
  name <> 'module' and
  name <> 'multiple' and
  name <> 'native' and
  name <> 'object' and
  name <> 'octet' and
  name <> 'oneway' and
  name <> 'out' and
  name <> 'primarykey' and
```

```
    name <> 'private' and
    name <> 'provides' and
    name <> 'public' and
    name <> 'publishes' and
    name <> 'raises' and
    name <> 'readonly' and
    name <> 'sequence' and
    name <> 'setraises' and
    name <> 'short' and
    name <> 'string' and
    name <> 'struct' and
    name <> 'supports' and
    name <> 'switch' and
    name <> 'true' and
    name <> 'truncatable' and
    name <> 'typedef' and
    name <> 'typeid' and
    name <> 'typeprefix' and
    name <> 'union' and
    name <> 'unsigned' and
    name <> 'uses' and
    name <> 'valuebase' and
    name <> 'valuetype' and
    name <> 'void' and
    name <> 'wchar' and
    name <> 'wstring'

  context IDLComposition
    /* A platform idl struct composition hierarchy must be consistent
     * with the composition hierarchy of the logical element
     * that it realizes. The platform value types must correspond
     * with the logical measurements and information elements.
     */
    inv idlStructConsistentWithLogical:
      let realizedComposition: face::logical::MeasurementComposition =
          self.realizes in
      let type: face::platform::ComposableElement = self.type in
      if type = null then
        false
      else
      if type.oclIsKindOf(face::platform::IDLPrimitive) then
        let idlType: face::platform::IDLPrimitive =
            type.oclAsType(face::platform::IDLPrimitive) in
        let logicalType: face::logical::ValueElement = idlType.realizes in
        logicalType = realizedComposition.type
      else
      if type.oclIsKindOf(face::platform::IDLStruct) then
        let idlType: face::platform::IDLStruct =
            type.oclAsType(face::platform::IDLStruct) in
        let logicalType: face::logical::ValueElement = idlType.realizes in
        logicalType = realizedComposition.type
      else
        false
      endif
      endif
      endif

endpackage
```

A.5 Constraints for *face::uop* Package

```
package face::uop
```

```
context Alias

/*
 * Ensure that alias name does not conflict with
 * a reserved word in IDL or FACE supported programming
 * language.
 */

  inv aliasNameNotReservedWord:
  let name: String = self.name.toLower() in
  name <> 'abstract' and
  name <> 'any' and
  name <> 'attribute' and
  name <> 'boolean' and
  name <> 'case' and
  name <> 'char' and
  name <> 'component' and
  name <> 'const' and
  name <> 'consumes' and
  name <> 'context' and
  name <> 'custom' and
  name <> 'default' and
  name <> 'double' and
  name <> 'emits' and
  name <> 'enum' and
  name <> 'eventtype' and
  name <> 'exception' and
  name <> 'factory' and
  name <> 'false' and
  name <> 'finder' and
  name <> 'fixed' and
  name <> 'float' and
  name <> 'getraises' and
  name <> 'home' and
  name <> 'import' and
  name <> 'in' and
  name <> 'inout' and
  name <> 'interface' and
  name <> 'local' and
  name <> 'long' and
  name <> 'manages' and
  name <> 'module' and
  name <> 'multiple' and
  name <> 'native' and
  name <> 'object' and
  name <> 'octet' and
  name <> 'oneway' and
  name <> 'out' and
  name <> 'primarykey' and
  name <> 'private' and
  name <> 'provides' and
  name <> 'public' and
  name <> 'publishes' and
  name <> 'raises' and
  name <> 'readonly' and
  name <> 'sequence' and
  name <> 'setraises' and
  name <> 'short' and
  name <> 'string' and
  name <> 'struct' and
  name <> 'supports' and
  name <> 'switch' and
```

```
    name <> 'true' and
    name <> 'truncatable' and
    name <> 'typedef' and
    name <> 'typeid' and
    name <> 'typeprefix' and
    name <> 'union' and
    name <> 'unsigned' and
    name <> 'uses' and
    name <> 'valuebase' and
    name <> 'valuetype' and
    name <> 'void' and
    name <> 'wchar' and
    name <> 'wstring'

endpackage
```

B FACE Technical Standard, Edition 2.1 Data Types

A review of FACE Technical Standard, Edition 2.1 data types suggests that the following data types are the most likely candidates to prevent public release:

- face.logical.Landmark
- face.logical.ReferencePoint
- face.logical.ReferencePointPart
- face.logical.MeasurementSystem
- face.logical.MeasurementSystemAxis
- face.logical.CoordinateSystem
- face.logical.CoordinateSystemAxis

B.1 Basis Elements

The following elements are the Basis Elements for FACE Edition 2.1:

- face.conceptual.Observable
- face.logical.Unit
- face.logical.Landmark
- face.logical.ReferencePoint
- face.logical.ReferencePointPart
- face.logical.MeasurementSystem
- face.logical.MeasurementSystemAxis
- face.logical.CoordinateSystem
- face.logical.CoordinateSystemAxis
- face.logical.MeasurementSystemConversion
- face.logical.Boolean
- face.logical.Character
- face.logical.Numeric
- face.logical.Integer
- face.logical.Natural
- face.logical.NonNegativeReal
- face.logical.Real
- face.logical.String

B.2 Constraint Helper Methods

```
package face

  context Element
    /*
     * 'tokenizeString' will return a sequence of strings that have been
     * split from the input string by the delimiter.
     * Assumes single character delimiter.
     */
    static def: tokenizeString(str : String, delimiter : String) : Sequence(String) =
      let i : Integer = str.indexOf(delimiter) in
      if i > 1
      then
          let token : String = str.substring(1, i-1) in
          if i = str.size()
          then
            Sequence{token}
          else
            let remainder : String = str.substring(i+1, str.size()) in
            let remainderTokens : Sequence(String) = tokenizeString(remainder,
              delimiter) in
            remainderTokens->prepend(token)
          endif
      else
        if i = str.size()
        then
          Sequence{}
        else
          if i = 1
          then
            let remainder : String = str.substring(i+1, str.size()) in
            self.tokenizeString(remainder, delimiter)
          else -- i <= 0
            Sequence{str}
          endif
        endif
      endif

    /*
     * Helper method to remove first node from a sequence of strings.
     */
    static def: removeFirstString(s : Sequence(String)) : Sequence(String) =
      if s->size() = 1 then s->select(false) else s->subSequence(2, s->size()) endif

    /*
     * Helper method to determine if string is a valid identifier.
     */
    static def: isValidIdentifier(str : String) : Boolean =
      str.size() > 0 and
      str.replaceAll('[_a-zA-Z][_a-zA-Z0-9]*', '').size() = 0 and
      not isReservedWord(str)

    /*
     * Helper method to determine if string is an valid format for a path.
     */
    static def: isValidPathFormat(str : String) : Boolean =
      str <> null implies

      --remove all entity projection substrings (e.g., ".description")
      let pathTmp1 = str.replaceAll('\\.[_a-zA-Z0-9]+', '') in

      --remove all entity projection substrings (e.g., "->description[A1]")
```

```
    let pathTmp2 = pathTmp1.replaceAll('->[_a-zA-Z0-9]+\\[[_a-zA-Z0-9]+\\]', '') in

    pathTmp2.size() = 0

  /*
   * Helper method to determine if string is an IDL reserved word.
   */
  static def: isReservedWord(str : String) : Boolean =
    let strLower: String = str.toLower() in
      strLower = 'abstract' or
      strLower = 'any' or
      strLower = 'attribute' or
      strLower = 'boolean' or
      strLower = 'case' or
      strLower = 'char' or
      strLower = 'component' or
      strLower = 'const' or
      strLower = 'consumes' or
      strLower = 'context' or
      strLower = 'custom' or
      strLower = 'default' or
      strLower = 'double' or
      strLower = 'emits' or
      strLower = 'enum' or
      strLower = 'eventtype' or
      strLower = 'exception' or
      strLower = 'factory' or
      strLower = 'false' or
      strLower = 'finder' or
      strLower = 'fixed' or
      strLower = 'float' or
      strLower = 'getraises' or
      strLower = 'home' or
      strLower = 'import' or
      strLower = 'in' or
      strLower = 'inout' or
      strLower = 'interface' or
      strLower = 'local' or
      strLower = 'long' or
      strLower = 'manages' or
      strLower = 'module' or
      strLower = 'multiple' or
      strLower = 'native' or
      strLower = 'object' or
      strLower = 'octet' or
      strLower = 'oneway' or
      strLower = 'out' or
      strLower = 'primarykey' or
      strLower = 'private' or
      strLower = 'provides' or
      strLower = 'public' or
      strLower = 'publishes' or
      strLower = 'raises' or
      strLower = 'readonly' or
      strLower = 'sequence' or
      strLower = 'setraises' or
      strLower = 'short' or
      strLower = 'string' or
      strLower = 'struct' or
      strLower = 'supports' or
      strLower = 'switch' or
      strLower = 'true' or
      strLower = 'truncatable' or
```

```
        strLower = 'typedef' or
        strLower = 'typeid' or
        strLower = 'typeprefix' or
        strLower = 'union' or
        strLower = 'unsigned' or
        strLower = 'uses' or
        strLower = 'valuebase' or
        strLower = 'valuetype' or
        strLower = 'void' or
        strLower = 'wchar' or
        strLower = 'wstring'

endpackage
```

B.3 Constraints for *face* Package

```
package face

  context Element
    /*
     * Check that name is a valid identifier.
     */
    inv nameIsValidIdentifier:
      not (self.oclIsTypeOf(face::conceptual::Generalization) or
           self.oclIsTypeOf(face::logical::Generalization) or
           self.oclIsTypeOf(face::platform::Generalization) or
           self.oclIsTypeOf(face::logical::Constraint))
      implies
        Element::isValidIdentifier(self.name)

  context ConceptualDataModel
    /*
     * Ensure that Generalization is in the same container as the specialized Entity.
     */
    inv conceptualGeneralizationSameContainerAsSpecialized:
      self.element
        ->select(e | e.oclIsTypeOf(face::conceptual::Generalization))
        ->collect(g | g.oclAsType(face::conceptual::Generalization))
        ->forAll(g | self.element->exists(e | e = g.specialized))

  context LogicalDataModel
    /*
     * Ensure that Generalization is in the same container as the specialized Entity.
     */
    inv logicalGeneralizationSameContainerAsSpecialized:
      self.element
        ->select(e | e.oclIsTypeOf(face::logical::Generalization))
        ->collect(g | g.oclAsType(face::logical::Generalization))
        ->forAll(g | self.element->exists(e | e = g.specialized))

  context PlatformDataModel
    /*
     * Ensure that Generalization is in the same container as the specialized Entity.
     */
    inv platformGeneralizationSameContainerAsSpecialized:
      self.element
        ->select(e | e.oclIsTypeOf(face::platform::Generalization))
        ->collect(g | g.oclAsType(face::platform::Generalization))
        ->forAll(g | self.element->exists(e | e = g.specialized))

endpackage
```

B.4 Constraints for *face::conceptual* Package

```
package face

  context Element
    /*
     * Get conceptual association by name.
     */
    static def: getConceptualAssociationByName(name : String)
     : face::conceptual::Association =
      let allAssociations : Collection(face::conceptual::Association) =
        face::conceptual::Association.allInstances() in
      -- there should be exactly one Association with expected name
      if allAssociations->one(a | a.name = name)
      then
        allAssociations->any(a | a.name = name)
      else
        null
      endif

    /*
     * Get composition path resolution.
     */
    static def: getConceptualAssociatedEntityFromToken( association :
face::conceptual::Association, token : String)
     : face::conceptual::Characteristic =

      let currPathTokenSplit : Sequence(String) =
          Element::tokenizeString(token.replaceAll(']', ''),'[') in
      let tokenRolename : String = currPathTokenSplit->first() in
      let allAssociations : Collection(face::conceptual::Association) =
        face::conceptual::Association.allInstances() in

      -- the Association should have exactly one AssociatedEntity with
      -- expected rolename
      if association.associatedEntity->one(c | c.rolename = tokenRolename)
      then
        association.associatedEntity->any(c | c.rolename = tokenRolename)
      else
        null
      endif

    /*
     * Get composition path resolution.
     */
    static def: getConceptualPathResolution(ce : face::conceptual::ComposableElement,
      token : String) : face::conceptual::Characteristic =

      if ce.oclIsKindOf(face::conceptual::Entity)
      then
        ce.oclAsType(face::conceptual::Entity).getCharacteristicByRolename(token)
      else
        null
      endif

    /*
     * Helper method to determine if a path relative to a ComposableElement is valid.
     */
    static def: resolveConceptualPath(ce : face::conceptual::ComposableElement,
pathTokens : Sequence(String)) : Sequence(face::conceptual::Characteristic) =
      if pathTokens->size() = 0
      then
        Sequence{}
```

```
      else
        let token : String = pathTokens->first() in
        if token.indexOf('[') > 0
        then
          let tokenSplit : Sequence(String) =
            Element::tokenizeString(token.replaceAll(']', ''),'[') in
          let rolename : String = tokenSplit->first() in
          let associationName : String = tokenSplit->last() in
          let association : face::conceptual::Association =
Element::getConceptualAssociationByName(associationName) in
            if association <> null
            then
              let resolvedCharacteristic : face::conceptual::Characteristic =
association.getCharacteristicByRolename(rolename) in
              if resolvedCharacteristic = null
              then
                Sequence{null}
              else
                Element::resolveConceptualPath(association,
Element::removeFirstString(pathTokens))
                  ->prepend(resolvedCharacteristic)
              endif
            else
              Sequence{null}
            endif
        else
          let resolvedCharacteristic : face::conceptual::Characteristic =
Element::getConceptualPathResolution(ce, token) in
          if resolvedCharacteristic = null
          then
            Sequence{null}
          else
            Element::resolveConceptualPath(resolvedCharacteristic.getType(),
Element::removeFirstString(pathTokens))
              ->prepend(resolvedCharacteristic)
          endif
        endif
      endif

endpackage

package face::conceptual

  context Element
    /*
     * Every face::conceptual::Element in the FACE data model shall
     * have a unique name.
     */
    inv hasUniqueName:
      let otherConceptualElements: Set(face::conceptual::Element) =
        face::conceptual::Element.allInstances()
          ->excluding(self)
          ->select(e | not e.oclIsTypeOf(face::conceptual::Generalization))
      in
      self.oclIsTypeOf(face::conceptual::Generalization) or
      not otherConceptualElements.name->exists(e | e = self.name)

  context Entity
    /*
     * Get characteristic by rolename.
     */
    def: getCharacteristicByRolename(rolename : String)
     : face::conceptual::Characteristic =
```

```
      let characteristics : Set(face::conceptual::Characteristic) =
self.getCharacteristics() in
      if characteristics->one(c | c.rolename = rolename)
      then
        characteristics->any(c | c.rolename = rolename)
      else
        null
      endif

    /*
     * A helper method that returns the Identity of an Entity.
     */
    def: getEntityIdentity() : Bag(OclAny) =
      self.getCharacteristics()->collect(c | c.getIdentityContribution())->asBag()

    /*
     * A helper method that returns the Characteristics of an Entity.
     */
    def: getCharacteristics() : OrderedSet(face::conceptual::Characteristic) =
      if self.oclIsTypeOf(face::conceptual::Association)
      then
        self.oclAsType(face::conceptual::Association)
          ->collect(associatedEntity.oclAsType(face::conceptual::Characteristic))
          ->union(self.composition)->asOrderedSet()
      else
        self.composition
      endif

    /*
     * The full composition hierarchy of each conceptual Entity shall be
     * unique. Uniqueness is determined by number, multiplicity and
     * type of its composed ComposableElement.
     */
    inv entityIsUnique:
      let ceIdentity: Bag(OclAny) =
        self.getEntityIdentity() in
        face::conceptual::Entity.allInstances()->excluding(self)
          ->forAll(ce | ce.getEntityIdentity() <> ceIdentity)

    /*
     * Every face::conceptual::Entity in the FACE data model shall contain
     * a face::conceptual::InformationElement named 'UniqueIDType'.
     */
    inv hasUniqueID:
      self.composition.type
        ->exists(a | a.oclIsTypeOf(Observable)
          and a.name = 'UniqueIdentifier'
        )

    /*
     * Every face::conceptual::Characteristic within the scope
     * of an Entity must have a unique name.
     */
    inv allCharacteristicsHaveUniqueRolename:
      self.getCharacteristics()->isUnique(rolename)

    /*
     * A conceptual Entity must be composed from conceptual
     * BasisElements or other conceptual Entities which
     * are composed of conceptual BasisElements.
     * This constraint ensures an Entity does not include
     * itself within its composition hierarchy.
     */
```

```
    inv entityConstructed:
      let ceClosure = self->closure(ce |
        let types = ce.composition.type->asSet()
          in
        let entityTypes = types
          ->select(t | t.oclIsTypeOf(face::conceptual::Entity)) in
        entityTypes->collect(t | t.oclAsType(face::conceptual::Entity))
      ) in
      not ceClosure->includes(self)

  context Generalization

    /*
     * A helper method that returns a bag of candidate identity contributions, one
     * candidate for each specializations of the input type.
    */
    def: candidateIdentityContributions(gc : Sequence(OclAny)) : Bag(OclAny) =
      let specializedType : face::conceptual::ComposableElement =
        gc->first().oclAsType(face::conceptual::ComposableElement) in
      let applicableGeneralizations : Set(face::conceptual::Generalization) =
        face::conceptual::Generalization.allInstances()
          ->select(gen | gen.specialized = specializedType) in
      let candidateReplacementIdentityContributions : Bag(Sequence(OclAny)) =
        applicableGeneralizations.generalized->collectNested(s | Sequence{s, gc-
>at(2), gc->at(3)}) in
      candidateReplacementIdentityContributions

    /*
     * Ensure that the specialized entity has a characteristic that corresponds to
     * each characteristic in the generalized entity. The corresponding specialized
     * characteristic may be of the same type or be a specialized type of the type
     * of the generalized characteristic.
     */
    inv generalizationStatementCorrect:
      let generalizedContents : Sequence(Sequence(OclAny)) =
        self.generalized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let specializedContents : Sequence(Sequence(OclAny)) =
        self.specialized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let generalizedContentCandidates : Bag(Sequence(OclAny)) =
        specializedContents->collectNested(gc : Sequence(OclAny) |
candidateIdentityContributions(gc))->iterate(
          replacementBag: Bag(Sequence(OclAny));
          acc : Bag(Sequence(OclAny)) = Bag{}|
          acc->union(replacementBag)
        )
      in

      generalizedContents->forAll(gc : Sequence(OclAny) | (
        specializedContents->exists(sc : Sequence(OclAny) | sc = gc) or
        generalizedContentCandidates->exists(scc : Sequence(OclAny) | scc = gc)
      ))

  context View

    /*
     * A helper method that returns the Identity of an Entity.
     */
    def: getViewIdentity() : Bag(OclAny) =
      self.characteristic->collect(c | c.getIdentityContribution())
```

```
    /*
     * The each View shall be unique.  Uniqueness is determined by
     * number, multiplicity, type, and context of its ProjectedCharacteristics.
     */
    inv viewIsUnique:
      let cvIdentity: Bag(OclAny) =
        self.getViewIdentity() in
        face::conceptual::View.allInstances()->excluding(self)
          ->forAll(cv | cv.getViewIdentity() <> cvIdentity
          )

    /*
     * Ensure that the rolename for each characteristic projection is unique
     * within a view. The rolename may be implicit or explicit.
     */
    inv rolenameIsUnique:
      let rolenames : Set(String) = self.characteristic->collect(c | c.getRolename())-
>asSet() in
      rolenames->forAll(rn | rn <> null) and rolenames->isUnique(rn | rn)

  context Characteristic

    def: getType()
     : face::conceptual::ComposableElement =
      if self.oclIsTypeOf(face::conceptual::Composition)
      then
        self.oclAsType(face::conceptual::Composition).type.oclAsType(face::
          conceptual::ComposableElement)
      else
        self.oclAsType(face::conceptual::AssociatedEntity).type.oclAsType(face::
          conceptual::ComposableElement)
      endif

    /*
     * A helper method that returns the contribution that
     * a Characteristic makes to an Entity's identity.
     */
    def: getIdentityContribution() : Sequence(OclAny) =
      Sequence{self.getType(), self.upperBound, self.lowerBound}

    /*
     * For conceptual, logical, and platform Compositions the lowerBound
     * shall be less than or equal to upperBound.
     */
    inv lowerBound_LTE_UpperBound:
      self.upperBound <> -1 implies self.lowerBound <= self.upperBound
    /*
     * For conceptual, logical, and platform Compositions, upperBound shall
     * be == -1 or >= 1.
     */
    inv upperBoundValid:
      self.upperBound = -1 or self.upperBound >= 1
    /*
     * For conceptual, logical, and platform Compositions, lowerBound shall
     * be >= zero.
     */
    inv lowerBoundValid:
      self.lowerBound  >= 0

    /*
     * Check that rolename is a valid identifier.
     */
```

```
  inv rolenameIsValidIdentifier:
    Element::isValidIdentifier(self.rolename)

context AssociatedEntity
  /*
   * AssociatedEntity must have a valid path.
   */
  inv associatedEntityPathValid:
    let ce : face::conceptual::ComposableElement =
      self.type.oclAsType(face::conceptual::ComposableElement) in
    let tokens : Sequence(String) =
      Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

    Element::isValidPathFormat(self.path) and
    Element::resolveConceptualPath(ce, tokens)->forAll(c | c <> null)

context CharacteristicProjection

  /*
   * CharacteristicProjection must have a valid path.
   */
  inv characteristicProjectionPathValid:
    let ce : face::conceptual::ComposableElement =
      self.projectedCharacteristic.oclAsType(face::conceptual::ComposableElement) in
    let tokens : Sequence(String) =
      Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

    Element::isValidPathFormat(self.path) and
    Element::resolveConceptualPath(ce, tokens)->forAll(c | c <> null)

  /*
   * A helper method that returns the contribution that
   * a Characteristic makes to an Entity's identity.
   */
  def: getIdentityContribution() : Sequence(OclAny) =
    Sequence{self.projectedCharacteristic, self.path}

  /*
   * A helper method that returns the computed rolename of a projection.
   */
  def: getRolename() : String =
    if self.rolename.size() > 0
    then
      self.rolename
    else
    if self.path.size() > 0 and
      self.path.substring(self.path.size(),self.path.size()) <> ']'
    then
      let ce : face::conceptual::ComposableElement =
        self.projectedCharacteristic.oclAsType(face::conceptual::ComposableElement)
        in
      let tokens : Sequence(String) =
        Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

      Element::resolveConceptualPath(ce, tokens)->last().rolename
    else
      null
    endif
    endif

  /*
   * If defined, check that rolename is a valid identifier.
   */
```

```
    inv rolenameIsValidIdentifier:
      self.rolename.size() > 0 implies
        Element::isValidIdentifier(self.rolename)

endpackage
```

B.5 Constraints for *face::logical* Package

```
package face

  context Element
    /*
     * Get logical association by name.
     */
    static def: getLogicalAssociationByName(name : String)
     : face::logical::Association =
      let allAssociations : Collection(face::logical::Association) =
        face::logical::Association.allInstances() in
      -- there should be exactly one Association with expected name
      if allAssociations->one(a | a.name = name)
      then
        allAssociations->any(a | a.name = name)
      else
        null
      endif

    /*
     * Get composition path resolution.
     */
    static def: getLogicalAssociatedEntityFromToken( association :
face::logical::Association, token : String)
     : face::logical::Characteristic =

      let currPathTokenSplit : Sequence(String) =
          Element::tokenizeString(token.replaceAll(']', ''),'[') in
      let tokenRolename : String = currPathTokenSplit->first() in
      let allAssociations : Collection(face::logical::Association) =
        face::logical::Association.allInstances() in

      -- the Association should have exactly one AssociatedEntity with expected
      -- rolename
      if association.associatedEntity->one(c | c.rolename = tokenRolename)
      then
        association.associatedEntity->any(c | c.rolename = tokenRolename)
      else
        null
      endif

    /*
     * Get composition path resolution.
     */
    static def: getLogicalPathResolution(ce : face::logical::ComposableElement, token
: String)
     : face::logical::Characteristic =

      if ce.oclIsKindOf(face::logical::Entity)
      then
        ce.oclAsType(face::logical::Entity).getCharacteristicByRolename(token)
      else
        null
      endif

    /*
```

```
     * Helper method to determine if a path relative to a ComposableElement is valid.
     */
    static def: resolveLogicalPath(ce : face::logical::ComposableElement, pathTokens :
Sequence(String)) : Sequence(face::logical::Characteristic) =
      if pathTokens->size() = 0
      then
        Sequence{}
      else
        let token : String = pathTokens->first() in
        if token.indexOf('[') > 0
        then
          let tokenSplit : Sequence(String) =
            Element::tokenizeString(token.replaceAll(']', ''),'[') in
          let rolename : String = tokenSplit->first() in
          let associationName : String = tokenSplit->last() in
          let association : face::logical::Association =
Element::getLogicalAssociationByName(associationName) in
            if association <> null
            then
              let resolvedCharacteristic : face::logical::Characteristic =
association.getCharacteristicByRolename(rolename) in
              if resolvedCharacteristic = null
              then
                Sequence{null}
              else
                Element::resolveLogicalPath(association,
Element::removeFirstString(pathTokens))
                    ->prepend(resolvedCharacteristic)
              endif
            else
              Sequence{null}
            endif
        else
          let resolvedCharacteristic : face::logical::Characteristic =
Element::getLogicalPathResolution(ce, token) in
          if resolvedCharacteristic = null
          then
            Sequence{null}
          else
            Element::resolveLogicalPath(resolvedCharacteristic.getType(),
Element::removeFirstString(pathTokens))
              ->prepend(resolvedCharacteristic)
          endif
        endif
      endif

endpackage

package face::logical

  context Element
    /*
     * Every face::logical::Element except Constraint and Generalization shall
     * have a unique name.
     */
    inv hasUniqueName:
      let otherLogicalElements: Set(face::logical::Element) =
        face::logical::Element.allInstances()
          ->excluding(self)
          ->select(e | not e.oclIsTypeOf(face::logical::Generalization))
          ->select(e | not e.oclIsTypeOf(face::logical::Constraint))
      in
      self.oclIsTypeOf(face::logical::Generalization) or
```

```
      self.oclIsTypeOf(face::logical::Constraint) or
      not otherLogicalElements.name->exists(e | e = self.name)

  context Entity
    /*
     * Get characteristic by rolename.
     */
    def: getCharacteristicByRolename(rolename : String)
     : face::logical::Characteristic =
      let characteristics : Set(face::logical::Characteristic) =
self.getCharacteristics() in
      if characteristics->one(c | c.rolename = rolename)
      then
        characteristics->any(c | c.rolename = rolename)
      else
        null
      endif

    /*
     * A helper method that returns the Characteristics of an Entity.
     */
    def: getCharacteristics() : OrderedSet(face::logical::Characteristic) =
      if self.oclIsTypeOf(face::logical::Association)
      then
        self.oclAsType(face::logical::Association)
          ->collect(associatedEntity.oclAsType(face::logical::Characteristic))
          ->union(self.composition)->asOrderedSet()
      else
        self.composition
      endif

    /*
     * Every face::logical::Characteristic within the scope
     * of an Entity must have a unique name.
     */
    inv allCharacteristicsHaveUniqueRolename:
      self.getCharacteristics()->isUnique(rolename)

    /*
     * Ensure that the Compositions in a logical Entity realize Compositions in the
     * conceptual Entity that the logical Entity realizes.
     */
    inv logicalEntityConsistentWithConceptual:
      self.composition.realizes->forAll(c | self.realizes.composition->exists(c2 | c =
c2))

  context Generalization

    /*
     * A helper method that returns a bag of candidate identity contributions, one
     * candidate for each specializations of the input type.
     */
    def: candidateIdentityContributions(gc : Sequence(OclAny)) : Bag(OclAny) =
      let specializedType : face::logical::ComposableElement =
        gc->first().oclAsType(face::logical::ComposableElement) in
      let applicableGeneralizations : Set(face::logical::Generalization) =
        face::logical::Generalization.allInstances()
          ->select(gen | gen.specialized = specializedType) in
      let candidateReplacementIdentityContributions : Bag(Sequence(OclAny)) =
        applicableGeneralizations.generalized->collectNested(s | Sequence{s, gc-
>at(2), gc->at(3)}) in
      candidateReplacementIdentityContributions
```

```
    /*
     * Ensure that the specialized entity has a characteristics that corresponds to
     * each characteristic in the generalized entity. The corresponding specialized
     * characteristic may be of the same type or be a specialized type of the type of
     * the generalized characteristic.
    */
    inv generalizationStatementCorrect:
      let generalizedContents : Sequence(Sequence(OclAny)) =
        self.generalized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let specializedContents : Sequence(Sequence(OclAny)) =
        self.specialized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let generalizedContentCandidates : Bag(Sequence(OclAny)) =
        specializedContents->collectNested(gc : Sequence(OclAny) |
candidateIdentityContributions(gc))->iterate(
          replacementBag: Bag(Sequence(OclAny));
          acc : Bag(Sequence(OclAny)) = Bag{}|
          acc->union(replacementBag)
        )
      in

      generalizedContents->forAll(gc : Sequence(OclAny) | (
        specializedContents->exists(sc : Sequence(OclAny) | sc = gc) or
        generalizedContentCandidates->exists(scc : Sequence(OclAny) | scc = gc)
      ))

  context Association

    /*
     * Ensure that the AssociatedEntities in a logical Association realize
     * AssociatedEntities in the conceptual Association that the logical
     * Association realizes.
     */
    inv logicalAssociationConsistentWithConceptual:
      let conceptualAssociationContents: Bag(face::conceptual::AssociatedEntity) =
        self.realizes.oclAsType(face::conceptual::Association).associatedEntity in
      self.associatedEntity.realizes->forAll(ae | conceptualAssociationContents-
>exists(ae2 | ae = ae2))

  context Characteristic
    /*
     * Helper method to get the type of a concrete Characteristic.
    */
    def: getType() : face::logical::ComposableElement =
      if self.oclIsTypeOf(face::logical::Composition)
      then
        self.oclAsType(face::logical::Composition).type.oclAsType(face::logical::
          ComposableElement)
      else
        self.oclAsType(face::logical::AssociatedEntity).type.oclAsType(face::
          logical::ComposableElement)
      endif

    /*
     * Helper method to get the realized characteristic of a concrete Characteristic.
     */
    def: getRealizes() : face::conceptual::Characteristic =
      if self.oclIsTypeOf(face::logical::Composition)
      then
        self.oclAsType(face::logical::Composition).realizes.oclAsType(face::
```

```
        conceptual::Characteristic)
    else
      self.oclAsType(face::logical::AssociatedEntity).realizes.oclAsType(face::
        conceptual::Characteristic)
    endif

  /*
   * A helper method that returns the contribution that
   * a Characteristic makes to an Entity's identity.
   */
  def: getIdentityContribution() : Sequence(OclAny) =
    Sequence{self.getType(), self.upperBound, self.lowerBound}

  /*
   * Check that rolename is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    Element::isValidIdentifier(self.rolename)

context Composition
  /*
   * Ensure that when an element realizes another element, the
   * upper and lower bounds of the realized entity match those
   * of the realizing entity.
   */
  inv logicalBoundsEqualConceptual:
    self.lowerBound = self.realizes.lowerBound and
    self.upperBound = self.realizes.upperBound

  /* A logical entity composition hierarchy must be consistent
   * with the composition hierarchy of the conceptual entity
   * that it realizes. The logical measurements must correspond
   * with the conceptual observables.
   */
  inv logicalCompositionConsistentWithConceptual:
    if self.type.oclIsKindOf(face::logical::Entity) then
      self.type.oclAsType(face::logical::Entity).realizes = self.realizes.type
    else
    if self.type.oclIsKindOf(face::logical::Measurement) then
      self.type.oclAsType(face::logical::Measurement).realizes = self.realizes.type
    else
      false
    endif
    endif

context AssociatedEntity

  /*
   * AssociatedEntity must have a valid path.
   */
  inv associatedEntityPathValid:
    let ce : face::logical::ComposableElement =
      self.type.oclAsType(face::logical::ComposableElement) in
    let tokens : Sequence(String) =
      Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

    Element::isValidPathFormat(self.path) and
    Element::resolveLogicalPath(ce, tokens)->forAll(c | c <> null)

  /*
   * The type of a logical AssociatedEntity must realize the same conceptual type
   * that is the type of the realized conceptual AssociatedEntity.
   */
```

```
  inv logicalAssociatedEntityConsistentWithConceptual:
    self.type.realizes = self.realizes.type

/*
 * Ensure that when an element realizes another element, the
 * upper and lower bounds of the realized entity match those
 * of the realizing entity.
 */
inv logicalBoundsEqualConceptual:
  self.lowerBound = self.realizes.lowerBound and
  self.upperBound = self.realizes.upperBound

context CharacteristicProjection

  /*
   * CharacteristicProjection must have a valid path.
   */
  inv characteristicProjectionPathValid:
    let ce : face::logical::ComposableElement =
      self.projectedCharacteristic.oclAsType(face::logical::ComposableElement) in
    let tokens : Sequence(String) =
      Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

    Element::isValidPathFormat(self.path) and
    Element::resolveLogicalPath(ce, tokens)->forAll(c | c <> null)

  /*
   * If a logical CharacteristicProjection realizes a conceptual
   * CharacteristicProjection the path must be follow the same path
   * as the conceptual.
   */
  inv logicalCharacteristicProjectionConsistentWithConcepual:
    self.realizes <> null implies

      let ce : face::logical::ComposableElement =
        self.projectedCharacteristic.oclAsType(face::logical::ComposableElement) in
      let tokens : Sequence(String) =
        Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in
      let logicalPath : Sequence(face::logical::Characteristic) =
        Element::resolveLogicalPath(ce, tokens) in
      let conceptualCE : face::conceptual::ComposableElement =
        self.realizes.projectedCharacteristic.oclAsType(face::
          conceptual::ComposableElement) in
      let conceptualTokens : Sequence(String) =
        Element::tokenizeString(self.realizes.path.replaceAll('->', '.'), '.') in
      let conceptualPath : Sequence(face::conceptual::Characteristic) =
        Element::resolveConceptualPath(conceptualCE, conceptualTokens) in

      logicalPath->collect(c | c.getRealizes()) = conceptualPath

  /*
   * If defined, check that rolename is a valid identifier.
   */
  inv rolenameIsValidIdentifier:
    self.rolename.size() > 0 implies
      Element::isValidIdentifier(self.rolename)

  /*
   * A helper method that returns the computed rolename of a projection.
   */
  def: getRolename() : String =
    if self.rolename.size() > 0
    then
```

```
        self.rolename
      else
      if self.path.size() > 0 and
        self.path.substring(self.path.size(),self.path.size()) <> ']'
      then
        let ce : face::logical::ComposableElement =
          self.projectedCharacteristic.oclAsType(face::logical::ComposableElement) in
        let tokens : Sequence(String) =
          Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

        Element::resolveLogicalPath(ce, tokens)->last().rolename
      else
        null
      endif
      endif

  context View
    /*
     * If a logical View realizes a conceptual View then all the
     * CharacteristicProjections should realize a conceptual
     * CharacteristicProjectin in the conceptual View.
     */
    inv logicalViewConsistentWithConceptual:
      if self.realizes = null
      then
        self.characteristic->forAll(c | c.realizes = null)
      else
        self.characteristic->forAll(c | self.realizes.characteristic->exists(c2 |
c.realizes = c2))
      endif

    /*
     * Ensure that the rolename for each characteristic projection is unique
     * within a view. The rolename may be implicit or explicit.
     */
    inv rolenameIsUnique:
      let rolenames : Set(String) = self.characteristic->collect(c | c.getRolename())-
>asSet() in
      rolenames->forAll(rn | rn <> null) and rolenames->isUnique(rn | rn)

  context ValueTypeUnit
    /*
     * An EnumeratedConstraint should select only labels from the Enumerated
     * instance it is constraining.
     */
    inv appropriateLabelsForEnumeratedConstraint:
      if self.constraint <> null
      then
        if self.constraint.oclIsTypeOf(face::logical::EnumerationConstraint)
        then
          self.valueType.oclIsTypeOf(face::logical::Enumerated) and
          self.constraint.oclAsType(face::
            logical::EnumerationConstraint).allowedValue->forAll(av |
            self.valueType.oclAsType(face::logical::Enumerated).label->exists(l | l =
av)
          )
        else
          true -- constraint is not an EnumerationConstraint
        endif
      else
        true -- with no constraint there is nothing to check
      endif
```

```
   context ValueType
     /*
     * The name of a ValueType instance should match the metaclass for except
     * for Enumerated ValueTypes.
     */
     inv nameOfValueTypeMatchesNameOfMetaclass:
       self.oclIsTypeOf(face::logical::Enumerated) or self.name = self.oclType().name

   context MeasurementSystem

     def: hasAnEnumeratedValueType() : Boolean =

       let valueTypes: Collection(face::logical::ValueType) =
self.measurementSystemAxis.defaultValueTypeUnit.valueType in

       valueTypes->exists(vt | vt.oclIsTypeOf(face::logical::Enumerated))

     /*
      * Only one measurement system with an Enumerated ValueType.
      */
     inv onlyOneEnumeratedMeasurementSystem:
       if self.name = 'AbstractDiscreteSetMeasurementSystem'
       then
         self.hasAnEnumeratedValueType() and
self.measurementSystemAxis.defaultValueTypeUnit->size() = 1
       else
         not self.hasAnEnumeratedValueType()
       endif

     /*
      * Ensure MeasurementSystemAxes have CoordinateSystemAxes that
      * are in the CoordinateSystem of the MeasurementSystem
      */
     inv measurementSystemConsistentWithCoordinateSystem:
       self.measurementSystemAxis.axis->asSet() = coordinateSystem.axis->asSet()

     /*
      * Ensure that ReferencePoint uses MeasurementSystemAxes and ValueTypeUnits
      * from the correct MeasurementSystem.
      */
     inv referencePointPartConsistentWithAxes:
       self.referencePoint.referencePointPart->forAll(rpp |
         rpp.axis->forAll(
           rppAxis | self.measurementSystemAxis->exists(msa | msa = rppAxis)
         ) and
         rpp.valueTypeUnit->forAll(
           rppVTU | self.measurementSystemAxis.defaultValueTypeUnit->exists(vtu | vtu
= rppVTU)
         )
       )

   context Measurement
     /*
      * Helper method to check if a Measurement has an Enumerated ValueType.
      */
     def: hasAnEnumeratedValueType() : Boolean =
       let valueTypes: Collection(face::logical::ValueType) =
             self.measurementAxis.valueTypeUnit.valueType in
       valueTypes->exists(vt | vt.oclIsTypeOf(face::logical::Enumerated))

     /*
      * Ensure that all Measurements with an Enumerated ValueType are associated
      * with the single 'AbstractDiscreteSetMeasurementSystem'.
```

```
      */
     inv allEnumeratedMeasurementUseEnumeratedMeasurementSystem:
       if self.hasAnEnumeratedValueType()
       then
         self.measurementSystem.name = 'AbstractDiscreteSetMeasurementSystem'
       else
         self.measurementSystem.name <> 'AbstractDiscreteSetMeasurementSystem'
       endif

     /*
      * Ensure MeasurementAxes have MeasurementSystemAxes that are in the
      * MeasurementSystem of the Measurement.
      */
     inv measurementConsistentWithMeasurementSystem:
       self.measurementAxis.measurementSystemAxis->asSet() =
measurementSystem.measurementSystemAxis->asSet()

endpackage
```

B.6 Constraints for *face::platform* Package

```
package face

  context Element
    /*
     * Get platform association by name.
     */
    static def: getPlatformAssociationByName(name : String)
     : face::platform::Association =
      let allAssociations : Collection(face::platform::Association) =
        face::platform::Association.allInstances() in
      -- there should be exactly one Association with expected name
      if allAssociations->one(a | a.name = name)
      then
        allAssociations->any(a | a.name = name)
      else
        null
      endif

    /*
     * Get composition path resolution.
     */
    static def: getPlatformAssociatedEntityFromToken( association :
face::platform::Association, token : String)
     : face::platform::Characteristic =

      let currPathTokenSplit : Sequence(String) =
          Element::tokenizeString(token.replaceAll(']', ''),'[') in
      let tokenRolename : String = currPathTokenSplit->first() in
      let allAssociations : Collection(face::platform::Association) =
        face::platform::Association.allInstances() in

      -- the Association should have exactly one AssociatedEntity with
      -- expected rolename
      if association.associatedEntity->one(c | c.rolename = tokenRolename)
      then
        association.associatedEntity->any(c | c.rolename = tokenRolename)
      else
        null
      endif

    /*
```

```
     * Get composition path resolution
     */
    static def: getPlatformPathResolution(ce : face::platform::ComposableElement,
token : String)
     : face::platform::Characteristic =

      if ce.oclIsKindOf(face::platform::Entity)
      then
        ce.oclAsType(face::platform::Entity).getCharacteristicByRolename(token)
      else
        null
      endif

    /*
     * Helper method to determine if a path relative to a ComposableElement is valid.
     */
    static def: resolvePlatformPath(ce : face::platform::ComposableElement, pathTokens
: Sequence(String)) : Sequence(face::platform::Characteristic) =
      if pathTokens->size() = 0
      then
        Sequence{}
      else
        let token : String = pathTokens->first() in
        if token.indexOf('[') > 0
        then
          let tokenSplit : Sequence(String) =
            Element::tokenizeString(token.replaceAll(']', ''),'[') in
          let rolename : String = tokenSplit->first() in
          let associationName : String = tokenSplit->last() in
          let association : face::platform::Association =
Element::getPlatformAssociationByName(associationName) in
            if association <> null
            then
              let resolvedCharacteristic : face::platform::Characteristic =
association.getCharacteristicByRolename(rolename) in
              if resolvedCharacteristic = null
              then
                Sequence{null}
              else
                Element::resolvePlatformPath(association,
Element::removeFirstString(pathTokens))
                    ->prepend(resolvedCharacteristic)
              endif
            else
              Sequence{null}
            endif
        else
          let resolvedCharacteristic : face::platform::Characteristic =
Element::getPlatformPathResolution(ce, token) in
          if resolvedCharacteristic = null
          then
            Sequence{null}
          else
            Element::resolvePlatformPath(resolvedCharacteristic.getType(),
Element::removeFirstString(pathTokens))
              ->prepend(resolvedCharacteristic)
          endif
        endif
      endif

endpackage

package face::platform
```

```
context Element
  /*
   * Every face::platform::Element except Generalization shall
   * have a unique name.
   */
  inv hasUniqueName:
    let otherPlatformElements: Set(face::platform::Element) =
      face::platform::Element.allInstances()
        ->excluding(self)
        ->select(e | not e.oclIsTypeOf(face::platform::Generalization))
    in
    self.oclIsTypeOf(face::platform::Generalization) or
    not otherPlatformElements.name->exists(e | e = self.name)

context Composition
  /*
   * Ensure that when an element realizes another element, the
   * upper and lower bounds of the realized entity match those
   * of the realizing entity.
   */
  inv platformBoundsEqualLogical:
    self.lowerBound = self.realizes.lowerBound and
    self.upperBound = self.realizes.upperBound

  /* A platform entity composition hierarchy must be consistent
   * with the composition hierarchy of the logical entity
   * that it realizes. The platform value types must correspond
   * with the logical measurements and information elements.
   */
  inv platformCompositionConsistentWithLogical:
    if self.type.oclIsKindOf(face::platform::Entity) then
      self.type.oclAsType(face::platform::Entity).realizes = self.realizes.type
    else
    if self.type.oclIsKindOf(face::platform::IDLType) then
      self.type.oclAsType(face::platform::IDLType).realizes = self.realizes.type
    else
      false
    endif
    endif

context Characteristic
  /*
   * Helper method to get the type of a concrete Characteristic.
   */
  def: getType()
   : face::platform::ComposableElement =
    if self.oclIsTypeOf(face::platform::Composition)
    then
      self.oclAsType(face::platform::Composition).type.oclAsType(face::
        platform::ComposableElement)
    else
      self.oclAsType(face::platform::AssociatedEntity).type.oclAsType(face::
        platform::ComposableElement)
    endif

  /*
   * Helper method to get the realized characteristic of a concrete Characteristic.
   */
  def: getRealizes() : face::logical::Characteristic =
    if self.oclIsTypeOf(face::platform::Composition)
    then
      self.oclAsType(face::platform::Composition).realizes.oclAsType(face::
```

```
          logical::Characteristic)
      else
        self.oclAsType(face::platform::AssociatedEntity).realizes.oclAsType(face::
          logical::Characteristic)
      endif

    /*
     * A helper method that returns the contribution that
     * a Characteristic makes to an Entity's identity.
     */
    def: getIdentityContribution() : Sequence(OclAny) =
      Sequence{self.getType(), self.upperBound, self.lowerBound}

    /*
     * Check that rolename is a valid identifier.
     */
    inv rolenameIsValidIdentifier:
      Element::isValidIdentifier(self.rolename)

  context Entity
    /*
     * Get characteristic by rolename.
     */
    def: getCharacteristicByRolename(rolename : String)
     : face::platform::Characteristic =
      let characteristics : Set(face::platform::Characteristic) =
self.getCharacteristics() in
      if characteristics->one(c | c.rolename = rolename)
      then
        characteristics->any(c | c.rolename = rolename)
      else
        null
      endif

    /*
     * A helper method that returns the Characteristics of an Entity.
     */
    def: getCharacteristics() : OrderedSet(face::platform::Characteristic) =
      if self.oclIsTypeOf(face::platform::Association)
      then
        self.oclAsType(face::platform::Association)
          ->collect(associatedEntity.oclAsType(face::platform::Characteristic))
          ->union(self.composition)->asOrderedSet()
      else
        self.composition
      endif

    /*
     * Every face::logical::Characteristic within the scope
     * of an Entity must have a unique name.
     */
    inv allCharacteristicsHaveUniqueRolename:
      self.getCharacteristics()->isUnique(rolename)

    /*
     * Ensure that the Compositions in a platform Entity realize Compositions in the
     * logical Entity that the platform Entity realizes.
     */
    inv platformEntityConsistentWithLogical:
      self.composition.realizes->forAll(c | self.realizes.composition->exists(c2 | c =
c2))

  context Association
```

```
    /*
     * Ensure that the AssociatedEntities in a platform Association realize
     * AssociatedEntities in the logical Association that the platform
     * Association realizes.
     */
    inv platformAssociationConsistentWithLogical:
      let logicalAssociationContents: Bag(face::logical::AssociatedEntity) =
        self.realizes.oclAsType(face::logical::Association).associatedEntity in
      self.associatedEntity.realizes->forAll(ae | logicalAssociationContents-
>exists(ae2 | ae = ae2))

  context Generalization

    /*
     * A helper method that returns a bag of candidate identity contributions, one
     * candidate for each specializations of the input type.
     */
    def: candidateIdentityContributions(gc : Sequence(OclAny)) : Bag(OclAny) =
      let specializedType : face::platform::ComposableElement =
        gc->first().oclAsType(face::platform::ComposableElement) in
      let applicableGeneralizations : Set(face::platform::Generalization) =
        face::platform::Generalization.allInstances()
          ->select(gen | gen.specialized = specializedType) in
      let candidateReplacementIdentityContributions : Bag(Sequence(OclAny)) =
        applicableGeneralizations.generalized->collectNested(s | Sequence{s, gc-
>at(2), gc->at(3)}) in
      candidateReplacementIdentityContributions

    /*
     * Ensure that the specialized entity has a characteristics that corresponds to
     * each characteristic in the generalized entity. The corresponding specialized
     * characteristic may be of the same type or be a specialized type of the
     * type of the generalized characteristic.
     */
    inv generalizationStatementCorrect:
      let generalizedContents : Sequence(Sequence(OclAny)) =
        self.generalized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let specializedContents : Sequence(Sequence(OclAny)) =
        self.specialized.getCharacteristics()->collectNested(c |
c.getIdentityContribution()) in

      let generalizedContentCandidates : Bag(Sequence(OclAny)) =
        specializedContents->collectNested(gc : Sequence(OclAny) |
candidateIdentityContributions(gc))->iterate(
          replacementBag: Bag(Sequence(OclAny));
          acc : Bag(Sequence(OclAny)) = Bag{}|
          acc->union(replacementBag)
        )
      in

      generalizedContents->forAll(gc : Sequence(OclAny) | (
        specializedContents->exists(sc : Sequence(OclAny) | sc = gc) or
        generalizedContentCandidates->exists(scc : Sequence(OclAny) | scc = gc)
      ))

  context AssociatedEntity
    /*
     * AssociatedEntity must have a valid path.
     */
    inv associatedEntityPathValid:
```

```
      let ce : face::platform::ComposableElement =
        self.type.oclAsType(face::platform::ComposableElement) in
      let tokens : Sequence(String) =
        Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

      Element::isValidPathFormat(self.path) and
      Element::resolvePlatformPath(ce, tokens)->forAll(c | c <> null)

    /*
     * The type of a platform AssociatedEntity must realize the same logical type
     * that is the type of the realized logical AssociatedEntity.
     */
    inv platformAssociatedEntityConsistentWithLogical:
      self.type.realizes = self.realizes.type

    /*
     * Ensure that when an element realizes another element, the
     * upper and lower bounds of the realized entity match those
     * of the realizing entity.
    */
    inv platformBoundsEqualLogical:
      self.lowerBound = self.realizes.lowerBound and
      self.upperBound = self.realizes.upperBound

  context View

    /*
     * If a platform View realizes a logical View then all the
     * CharacteristicProjections should realize a logical
     * CharacteristicProjectin in the logical View.
     */
    inv platformViewConsistentWithLogical:
      if self.realizes = null
      then
        self.characteristic->forAll(c | c.realizes = null)
      else
        self.characteristic->forAll(c | self.realizes.characteristic->exists(c2 |
c.realizes = c2))
      endif

    /*
     * Ensure that the rolename for each characteristic projection is unique
     * within a view. The rolename may be implicit or explicit.
     */
    inv rolenameIsUnique:
      let rolenames : Set(String) = self.characteristic->collect(c | c.getRolename())-
>asSet() in
      rolenames->forAll(rn | rn <> null) and rolenames->isUnique(rn | rn)

  context CharacteristicProjection

    /*
     * CharacteristicProjection must have a valid path.
     */
    inv characteristicProjectionPathValid:
      let ce : face::platform::ComposableElement =
        self.projectedCharacteristic.oclAsType(face::platform::ComposableElement) in
      let tokens : Sequence(String) =
        Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

      Element::isValidPathFormat(self.path) and
      Element::resolvePlatformPath(ce, tokens)->forAll(c | c <> null)
```

```
    /*
     * If a platform CharacteristicProjection realizes a logical
     * CharacteristicProjection the path must be follow the same path as the logical.
     */
    inv platformCharacteristicProjectionConsistentWithLogical:
      self.realizes <> null implies

        let ce : face::platform::ComposableElement =
          self.projectedCharacteristic.oclAsType(face::platform::ComposableElement) in
        let tokens : Sequence(String) =
          Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in
        let platformPath : Sequence(face::platform::Characteristic) =
          Element::resolvePlatformPath(ce, tokens) in
        let logicalCE : face::logical::ComposableElement =
          self.realizes.projectedCharacteristic.oclAsType(face::
            logical::ComposableElement) in
        let logicalTokens : Sequence(String) =
          Element::tokenizeString(self.realizes.path.replaceAll('->', '.'), '.') in
        let logicalPath : Sequence(face::logical::Characteristic) =
          Element::resolveLogicalPath(logicalCE, logicalTokens) in

        platformPath->collect(c | c.getRealizes()) = logicalPath

    /*
     * If defined, check that rolename is a valid identifier.
     */
    inv rolenameIsValidIdentifier:
      self.rolename.size() > 0 implies
        Element::isValidIdentifier(self.rolename)

    /*
     * A helper method that returns the computed rolename of a projection.
     */
    def: getRolename() : String =
      if self.rolename.size() > 0
      then
        self.rolename
      else
      if self.path.size() > 0 and
        self.path.substring(self.path.size(),self.path.size()) <> ']'
      then
        let ce : face::platform::ComposableElement =
          self.projectedCharacteristic.oclAsType(face::platform::ComposableElement) in
        let tokens : Sequence(String) =
          Element::tokenizeString(self.path.replaceAll('->', '.'), '.') in

        Element::resolvePlatformPath(ce, tokens)->last().rolename
      else
        null
      endif
      endif

  context IDLType

    /*
     * A helper method that returns the ValueTypeUnit collection for a
     * MeasurementAxis.
     */
    static def: getValueUnitTypes(measurementAxis : face::logical::MeasurementAxis)
     : OrderedSet(face::logical::ValueTypeUnit) =

      let measurementAxisContents: OrderedSet(face::logical::ValueTypeUnit) =
measurementAxis.valueTypeUnit in
```

```
      if measurementAxisContents->size() > 0
      then
        -- return the overridden ValueTypeUnit collection of the MeasurementAxis
        measurementAxisContents
      else
        -- return the default ValueTypeUnit collection since there is no override
        let measurementSystemAxis: face::logical::MeasurementSystemAxis =
          measurementAxis.measurementSystemAxis in
        measurementSystemAxis.defaultValueTypeUnit
      endif

  context IDLStruct
    /*
     *  An IDL struct cannot realize a logical value type unit.
     */
    inv idlStructDoesNotRealizeValueTypeUnit:
      not self.realizes.oclIsTypeOf(face::logical::ValueTypeUnit)

    inv idlStructRealizesMultiPartMeasurement:
      -- check that realization is to a multi-part measurement
      let abstractMeasurement: face::logical::AbstractMeasurement = self.realizes in
      if abstractMeasurement.oclIsTypeOf(face::logical::Measurement)
      then
        let measurement : face::logical::Measurement =
          abstractMeasurement.oclAsType(face::logical::Measurement) in
        let measurementAxes : Sequence(face::logical::MeasurementAxis) =
measurement.measurementAxis in
        if measurementAxes->size() > 1
        then
          -- the IDL struct realizes a measurement with more than one axis
          true
        else
          -- if the measurement has a single axis ensure that the axis is multi part
          let measurementAxisContents: Sequence(face::logical::ValueTypeUnit) =
            IDLType::getValueUnitTypes(measurementAxes->first()) in
          measurementAxisContents->size() > 1
        endif
      else
      if abstractMeasurement.oclIsTypeOf(face::logical::MeasurementAxis)
      then
        -- ensure that the axis is multi part
        let measurementAxisContents: Sequence(face::logical::ValueTypeUnit) =
          IDLType::getValueUnitTypes(abstractMeasurement.oclAsType(face::
            logical::MeasurementAxis)) in
        measurementAxisContents->size() > 1
      else
        -- abstractMeasurement is a ValueTypeUnit which is atomic
        false
      endif
      endif

    /*
     * Ensure that the contents of an IDL struct realize the contents of the
     * measurement or measurement axis which the struct realizes.
     */
    inv idlStructConsistentWithLogical:
      -- check that struct is consistent with the logical model
      let containedTypeRealizations: Collection(face::logical::AbstractMeasurement) =
        self.composition.type.realizes in
      let abstractMeasurementContainedParts:
OrderedSet(face::logical::AbstractMeasurement) =

        if self.realizes.oclIsTypeOf(face::logical::Measurement)
```

```
        then
          let measurement : face::logical::Measurement =
            self.realizes.oclAsType(face::logical::Measurement) in
          let measurementAxes : Sequence(face::logical::MeasurementAxis) =
measurement.measurementAxis in
          if measurementAxes->size() > 1
          then
            measurementAxes->collect(c |
c.oclAsType(face::logical::AbstractMeasurement))->asSet()
          else
            IDLType::getValueUnitTypes(measurementAxes-
>any(true).oclAsType(face::logical::MeasurementAxis))
          endif
        else
        if self.realizes.oclIsTypeOf(face::logical::MeasurementAxis)
        then
          IDLType::getValueUnitTypes(self.realizes.oclAsType(face::
            logical::MeasurementAxis))
        else
          -- ValueTypeUnit cannot contain anything
          OrderedSet{}
        endif
        endif
      in

      abstractMeasurementContainedParts->asSet() = containedTypeRealizations->asSet()

  context IDLPrimitive

    /*
     *  An IDL primitive can only realize an abstract
     *  measurement that has a single value type unit.
     */
    inv idlPrimitiveRealizesAtomicAbstractMeasurement:
      -- check that realization is to an atomic measurement
      let abstractMeasurement: face::logical::AbstractMeasurement = self.realizes in
      if abstractMeasurement.oclIsTypeOf(face::logical::Measurement)
      then
        let measurement : face::logical::Measurement =
          abstractMeasurement.oclAsType(face::logical::Measurement) in
        let measurementAxes : Sequence(face::logical::MeasurementAxis) =
measurement.measurementAxis in
        if measurementAxes->size() > 1
        then
          -- the IDL primitive should not realize a measurement with more than one
          -- axis
          false
        else
          -- if the measurement has a single axis ensure that the axis is
          -- not multi part
          let measurementAxisContents: Sequence(face::logical::ValueTypeUnit) =
            IDLType::getValueUnitTypes(measurementAxes->first()) in
          measurementAxisContents->size() <= 1
        endif
      else
      if abstractMeasurement.oclIsTypeOf(face::logical::MeasurementAxis)
      then
        -- ensure that the axis is not multi part
        let measurementAxisContents: Sequence(face::logical::ValueTypeUnit) =
          IDLType::getValueUnitTypes(abstractMeasurement.oclAsType(face::
            logical::MeasurementAxis)) in
        measurementAxisContents->size() <= 1
      else
```

```
        -- abstractMeasurement must be a ValueTypeUnit which is atomic
        true
      endif
      endif

endpackage
```

C FACE Technical Standard, Edition 3.0 Data Types

A review of FACE Technical Standard, Edition 3.0 data types suggests that the following data types are the most likely candidates to prevent public release:

- face.logical.Landmark
- face.logical.ReferencePoint
- face.logical.ReferencePointPart
- face.logical.MeasurementSystem
- face.logical.MeasurementSystemAxis
- face.logical.CoordinateSystem
- face.logical.CoordinateSystemAxis

C.1 Basis Elements

The following elements are the Basis Elements for FACE Edition 3.0:

- face.datamodel.conceptual.Observable
- face.datamodel.conceptual.Domain
- face.datamodel.conceptual.BasisEntity
- face.datamodel.logical.Unit
- face.datamodel.logical.Landmark
- face.datamodel.logical.ReferencePoint
- face.datamodel.logical.ReferencePointPart
- face.datamodel.logical.StandardMeasurementSystem
- face.datamodel.logical.MeasurementSystem
- face.datamodel.logical.MeasurementSystemAxis
- face.datamodel.logical.CoordinateSystem
- face.datamodel.logical.CoordinateSystemAxis
- face.datamodel.logical.MeasurementSystemConversion
- face.datamodel.logical.Boolean
- face.datamodel.logical.Character
- face.datamodel.logical.Numeric
- face.datamodel.logical.Integer
- face.datamodel.logical.Natural
- face.datamodel.logical.NonNegativeReal

- face.datamodel.logical.Real
- face.datamodel.logical.String

C.2 Query Rules

Legend

term	The name of a term in the Query grammar. A set of rules may follow a term. These rules apply to all instances of this term in a Query. This term is referred to as the rule's "context term" in this legend.
term_name	The name of a term in the Query grammar. In a rule, it represents some instance of the named term in a Query. (Rules reference the context term as well as other terms in its expression.)
DATAMODELMETATYPE_NAME	The name of a DataModel metatype. In a rule, it represents some instance of the named metatype in a DataModel.
property_name	The name of DataModel metatype property. In a rule, it represents a value associated with the named property of some instance of a DataModel metatype in a DataModel.
"literal"	Represents a literal value.

query_specification

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a CONCEPTUALDATAMODEL, then:

- An ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a CONCEPTUALDATAMODEL
- An ***enum_literal_reference_expression*** must not be specified
- An ***enum_literal_set*** must not be specified

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a LOGICALDATAMODEL, then an ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a LOGICALDATAMODEL.

If the QUERY whose **specification** is this ***query_specification*** is an **element** of a PLATFORMDATAMODEL, then an ***entity_type_reference*** must match the **name** of one and only one ENTITY that is an **element** of a PLATFORMDATAMODEL.

query_statement

A ***selected_entity_reference*** in a ***projected_characteristic_expression*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***.

If an ***explicit_selected_entity_characteristic_reference*** is specified, and ***selected_entity_reference*** is not specified in its ***selected_entity_characteristic_reference***, then its ***characteristic_reference*** must match

the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***.

If a ***where_clause*** is specified, then:

- There must be at least one ***boolean_predicate*** in either its ***criteria*** or in a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***query_statement*** is not a ***subquery***, then for each ***selected_entity_characteristic_reference predicate_term*** in the ***where_clause***'s ***criteria***:
  - — If a ***selected_entity_reference*** is specified, then the ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***
  - — If a ***selected_entity_reference*** is not specified, then the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***query_statement*** is a ***subquery***, then for each ***selected_entity_characteristic_reference predicate_term*** in the ***where_clause***'s ***criteria***:
  - — If a ***selected_entity_reference*** is specified, then the ***selected_entity_reference*** must either:
    - Match by name a ***selected_entity_alias*** in the ***subquery***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***subquery***'s ***entity_expression***, or
    - Match by name a ***selected_entity_alias*** in an outer ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in an outer ***query_statement***'s ***entity_expression***

    If both □—□ and □—□ above are true, then the ***selected_entity*** from □—□ is assumed.
  - — If a ***selected_entity_reference*** is not specified, then the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in either:
    - The ***subquery***'s ***entity_expression***, or
    - An outer ***query_statement***'s ***entity_expression***

    If both □ and □ above are true, then the CHARACTERISTIC from □ is assumed.
- If its ***criteria*** contains a ***scalar_compare_predicate***, then:
  - — If both ***predicate_term***s are a ***selected_entity_characteristic_reference***, then at least one ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
  - — If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is an ***enum_literal_reference_expression***, then the ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
  - — If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is a ***subquery***, then either the ***selected_entity_characteristic_reference***

must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If one ***predicate_term*** is an ***enum_literal_reference_expression*** and the other ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If both ***predicate_term***s are a ***subquery***, then there must be at least one ***boolean_predicate*** in either ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***set_compare_predicate***, then:

— If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is an ***enum_literal_reference_expression***, then there must be at least one ***boolean_predicate*** in either the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either that ***subquery***'s ***where_clause*** or the ***compare_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***set_membership_predicate***, then:

— If ***logical_set*** is a ***subquery***, then:

▪ If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

▪ If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or the ***subquery predicate_term***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is

a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If ***predicate_term*** is a ***enum_literal_reference_expression***, then there must be at least one ***boolean_predicate*** in either the ***logical_set subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***logical_set*** is a ***characteristic_basis_set***, then:

- If a ***characteristic_basis*** in ***characteristic_basis_set*** is a ***selected_entity_characteristic_reference***, then that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then either that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***selected_entity_characteristic_reference characteristic_basis*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either that ***subquery***'s ***where_clause*** or a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***predicate_term*** is a ***enum_literal_reference_expression***, then there must be at least one ***selected_entity_characteristic_reference characteristic_basis*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***, or there must be at least one ***boolean_predicate*** in either a ***subquery characteristic_basis***' ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

— If ***logical_set*** is a ***enum_literal_set***, then:

- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then that ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***
- If ***predicate_term*** is a ***subquery***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a

CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

- If its ***criteria*** contains a ***exists_predicate***, then there must be at least one ***boolean_predicate*** in either the ***subquery***'s ***where_clause*** or a nested ***subquery***'s ***where_clause*** that contains a ***selected_entity_characteristic_reference predicate_term*** that is a CHARACTERISTIC of a ***selected_entity*** in the ***query_statement***'s ***entity_expression***

If an ***order_by_clause*** is specified, then for each ***ordering_expression***:

- If a ***qualified_projected_characteristic_reference*** is specified, then:
  - Its ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***query_statement***'s ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***query_statement***'s ***entity_expression***
  - Its ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC in the ***selected_entity*** referenced by ***selected_entity_reference***
  - It must be a CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***
- If an ***unqualified_projected_characteristic_reference_or_alias*** is specified, then:
  - It either must match by name a ***projected_characteristic_alias***, or it must match the **rolename** of one and only one CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***; if it matches by name a ***projected_characteristic_alias*** and also matches the **rolename** of one and only one CHARACTERISTIC in the ***projected_characteristic_list***, then the CHARACTERISTIC associated with the ***projected_characteristic_alias*** is assumed
- Two or more ***projected_characteristic_reference***s must not reference the same CHARACTERISTIC in the ***query_statement***'s ***projected_characteristic_list***

set_qualifier

// No contextually relevant rules for instances of this term.

projected_characteristic_list

A ***projected_characteristic_list*** must have at least one CHARACTERISTIC.

An ***explicit_selected_entity_characteristic_reference*** must not be a CHARACTERISTIC of a ***selected_entity*** referenced by a ***selected_entity_characteristic_wildcard_reference***.

Two or more ***selected_entity_characteristic_wildcard_reference***s must not reference the same ***selected_entity***.

Two or more ***explicit_selected_entity_characteristic_reference***s must not reference the same CHARACTERISTIC.

All ***projected_entity_alias***es must be unique.

all_characteristics

// No contextually relevant rules for instances of this term.

projected_characteristic_expression

// No contextually relevant rules for instances of this term.

selected_entity_characteristic_wildcard_reference

The ***selected_entity_reference*** must be a ***selected_entity*** with at least one CHARACTERISTIC whose **type** is not an ENTITY.

explicit_selected_entity_characteristic_reference

The ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC whose **type** is not an ENTITY.

selected_entity_expression

// No contextually relevant rules for instances of this term.

from_clause

// No contextually relevant rules for instances of this term.

entity_expression

All ***selected_entity_alias***es must be unique.

A ***selected_entity_alias*** must not be the same as any ***selected_entity***'s ***entity_type_reference***.

A ***selected_entity_alias*** must be specified for a ***selected_entity*** if there is more than one ***selected_entity*** with the same ***entity_type_reference***.

A ***selected_entity_reference*** must match by name a ***selected_entity_alias*** in the ***entity_expression*** or the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***entity_expression***.

If ***selected_entity_reference*** is not specified in a ***selected_entity_characteristic_reference***, then the ***selected_entity_characteristic_reference***'s ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC of one and only one ***selected_entity*** in the ***entity_expression***.

If an ***entity_expression*** has N ***selected_entity***s, where N > 1, then there must at least (N – 1) ***selected_entity_characteristic_reference***s whose **type**s are (N – 1) of the ***selected_entity***s in ***entity_expression***.

Two or more ***selected_entity_characteristic_reference***s must not be the same CHARACTERISTIC.

If an ***entity_join_expression*** is specified, then for each ***entity_type_characteristic_equivalence_expression*** in that ***entity_join_expression***:

- If its ***selected_entity_characteristic_reference*** is a CHARACTERISTIC of ***join_entity***, then:
    - — If a ***selected_entity_reference*** is specified after the ***equals_operator***, then the CHARACTERISTIC's **type**'s **name** must match by name the ***selected_entity_reference***'s ***entity_type_reference***, and the referenced ***selected_entity*** must not be ***join_entity***

— Otherwise, that CHARACTERISTIC's **type**'s **name** must match by name the ***entity_type_reference*** of one and only one ***selected_entity*** in the ***entity_expression***, and that ***selected_entity*** must not be ***join_entity***

- Otherwise, the ***selected_entity_characteristic_reference*** must not be a CHARACTERISTIC of ***join_entity***, and that CHARACTERISTIC's **type**'s **name** must match by name ***join_entity***'s ***entity_type_reference***; if ***selected_entity_reference*** is specified after the ***equals_operator***, then the referenced ***selected_entity*** must be ***join_entity***

selected_entity

A ***selected_entity_alias***, if specified, must not be the same as the ***entity_type_reference***.

entity_join_expression

// No contextually relevant rules for instances of this term.

join_entity

// No contextually relevant rules for instances of this term.

entity_join_criteria

// No contextually relevant rules for instances of this term.

entity_type_characteristic_equivalence_expression

The ***selected_entity_characteristic_reference*** must be a CHARACTERISTIC whose **type** is an ENTITY.

selected_entity_characteristic_reference

If ***selected_entity_reference*** is specified, the ***characteristic_reference*** must match the **rolename** of one and only one CHARACTERISTIC in the ***selected_entity*** referenced by ***selected_entity_reference***.

selected_entity_reference

// No contextually relevant rules for instances of this term.

where_clause

// No contextually relevant rules for instances of this term.

criteria

A ***selected_entity_characteristic_reference predicate_term*** must not reference a CHARACTERISTIC of a ***selected_entity*** in a nested ***subquery***'s ***entity_expression***.

order_by_clause

// No contextually relevant rules for instances of this term.

ordering_expression

// No contextually relevant rules for instances of this term.

projected_characteristic_reference

// No contextually relevant rules for instances of this term.

qualified_projected_characteristic_reference

// No contextually relevant rules for instances of this term.

unqualified_projected_characteristic_reference_or_alias

// No contextually relevant rules for instances of this term.

ordering_type

// No contextually relevant rules for instances of this term.

boolean_expression

// No contextually relevant rules for instances of this term.

boolean_term

// No contextually relevant rules for instances of this term.

boolean_factor

// No contextually relevant rules for instances of this term.

boolean_predicate

// No contextually relevant rules for instances of this term.

scalar_compare_predicate

Both ***predicate_term***s must not be ***enum_literal_reference_expression***s.

If both ***predicate_term***s are a ***selected_entity_characteristic_reference***, then both ***selected_entity_characteristic_reference***s must be a CHARACTERISTIC whose **type**s are the same.

If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is an ***enum_literal_reference_expression***, then:

- The ***selected_entity_characteristic_reference*** is a CHARACTERISTIC whose **type** must either be or realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*

- The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in □; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT
- The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***

If one ***predicate_term*** is a ***selected_entity_characteristic_reference*** and the other ***predicate_term*** is a ***subquery***, then the ***selected_entity_characteristic_reference***'s **type** and the ***subquery***'s projected CHARACTERISTIC's **type** must be the same.

If one ***predicate_term*** is an ***enum_literal_reference_expression*** and the other ***predicate_term*** is a ***subquery***, then:

- The ***subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*
- The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in □; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT
- The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***

If both ***predicate_term***s are a ***subquery***, then the projected CHARACTERISTIC's **type** in both ***subquery***s must be the same.

set_membership_predicate

If ***logical_set*** is a ***subquery***, then:

- There must be one and only one CHARACTERISTIC in its ***projected_characteristic_list***, and all instances of that CHARACTERISTIC's associated data must be scalar
- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then its **type** and the ***subquery***'s projected CHARACTERISTIC's **type** must be the same
- If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***logical_set subquery***'s projected CHARACTERISTIC's **type**
- If ***predicate_term*** is an ***enum_literal_reference_expression***, then:
  - The ***logical_set subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*
  - The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in □—; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT

If ***logical_set*** is a ***characteristic_basis_set***, then:

- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then its **type** and the ***characteristic_basis_set***'s **type** must be the same

- If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***characteristic_basis_set***'s **type**
- If ***predicate_term*** is an ***enum_literal_reference_expression***, then:
    - The ***characteristic_basis_set***'s **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is "*AbstractDiscreteSetMeasurementSystem"*
    - The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in —; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT

If ***logical_set*** is an ***enum_literal_set***, then:

- If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then:
    - The ***selected_entity_characteristic_reference*** is a CHARACTERISTIC whose **type** must either be or realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*
    - The ***enum_literal_set***'s ***enumeration_type_reference*** must match the **name** of the ENUMERATED in the MEASUREMENT defined in —
    - Each ***enumeration_literal_reference*** in ***enum_literal_set*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED defined in —; if an ENUMERATIONCONSTRAINT is specified for th MEASUREMENT defined in —, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT
- If ***predicate_term*** is a ***subquery***, then:
    - The ***subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*
    - The ***enum_literal_set***'s ***enumeration_type_reference*** must match the **name** of the ENUMERATED in the MEASUREMENT defined in —
    - Each ***enumeration_literal_reference*** in ***enum_literal_set*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED defined in —; if an ENUMERATIONCONSTRAINT is specified for th MEASUREMENT defined in —, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT
- ***predicate_term*** must not be an ***enum_literal_reference_expression***

logical_set

// No contextually relevant rules for instances of this term.

characteristic_basis_set

All of the characteristic_basis' **type**s must be the same.

set_compare_predicate

There must be one and only one CHARACTERISTIC in that ***compare_set subquery***'s ***projected_characteristic_list***, and all instances of that CHARACTERISTIC's associated data must be scalar.

If ***predicate_term*** is a ***selected_entity_characteristic_reference***, then it must be a CHARACTERISTIC whose **type** is the same as the ***compare_set subquery***'s projected CHARACTERISTIC's **type**.

If ***predicate_term*** is a ***subquery***, then that ***subquery***'s projected CHARACTERISTIC's **type** must be the same as the ***compare_set subquery***'s projected CHARACTERISTIC's **type.**

If ***predicate_term*** is an ***enum_literal_reference_expression***:

- The ***compare_set subquery***'s projected CHARACTERISTIC's **type** either must be or must realize a MEASUREMENT whose **measurementSystem** is a MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*
- The ***enum_literal_reference_expression*** is an ENUMERATIONLABEL that is a **label** of the ENUMERATED for the MEASUREMENT defined in □; if an ENUMERATIONCONSTRAINT is specified for that MEASUREMENT, then the ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT
- The ***compare_operator*** must be either ***equals_operator*** or ***not_equals_operator***

compare_set

// No contextually relevant rules for instances of this term.

compare_operator

// No contextually relevant rules for instances of this term.

set_compare_quantifier

// No contextually relevant rules for instances of this term.

exists_predicate

The ***subquery***'s ***projected_characteristic_list*** must be ***all_characteristics***.

predicate_term

// No contextually relevant rules for instances of this term.

characteristic_basis

If a ***characteristic_basis*** is a ***selected_entity_characteristic_reference***, then it must be a CHARACTERISTIC whose:

- **type** is not an ENTITY
- **lowerBound** and **upperBound** is 1
- Associated data is scalar

If a ***characteristic_basis*** is a ***subquery***, then:

- There must be one and only one CHARACTERISTIC in its ***projected_characteristic_list***

- That CHARACTERISTIC's l**owerBound** and **upperBound** must be 1
- That CHARACTERISTIC's associated data must be scalar

subquery

An ***order_by_clause*** must not be specified.

characteristic_reference

// No contextually relevant rules for instances of this term.

entity_type_reference

// No contextually relevant rules for instances of this term.

enum_literal_set

Each ***enumeration_literal_reference*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED whose **name** is ***enumeration_type_reference***.

All ***enumeration_literal_reference***s must be unique.

enum_literal_reference_expression

The ***enumeration_literal_reference*** must match the **name** of an ENUMERATIONLABEL that is a **label** of the ENUMERATED whose **name** is ***enumeration_type_reference***.

enumeration_type_reference

// No contextually relevant rules for instances of this term.

enumeration_literal_reference

// No contextually relevant rules for instances of this term.

selected_entity_alias

// No contextually relevant rules for instances of this term.

projected_characteristic_alias

// No contextually relevant rules for instances of this term.

query_identifier

A ***query_identifier*** is a valid String as defined by OCL constraint face::Element::isValidIdentifier specified in the FACE Technical Standard §J.6.1: OCL Constraint Helper Methods.

C.3 Template Rules

Legend

term	The name of a term in the Template grammar. A set of rules may follow a term. These rules apply to all instances of this term in a modeled TEMPLATE. This term is referred to as the rule's "context term" in this legend.
template_term_name	The name of a term in the Template grammar. In a rule, it represents an instance of the named term in a modeled TEMPLATE. (Rules reference the context term as well as other terms in its expression.)
DATAMODELMETATYPE_NAME	The name of a DataModel metatype. In a rule, it represents some instance of the named metatype in a modeled DATAMODEL.
QUERY_TERM_NAME	The name of a term in the Query grammar. In a rule, it represents an instance of the named term in the **specification** of the QUERY (which itself is a ***QUERY_SPECIFICATION***) that is the **boundQuery** of the TEMPLATE whose **specification** is the ***template_specification*** to which the rules below are being applied.
property_name	The name of DataModel metatype property. In a rule, it represents a value associated with the named property of some instance of a metatype in a modeled DATAMODEL.
"literal"	Represents a literal value.

template_specification

There must be one and only one ***main_template_method_decl***, either a ***main_template_method_decl*** or a ***main_equivalent_entity_type_template_method_decl***.

If ***main_template_method_decl*** is a ***main_equivalent_entity_type_template_method_decl***, then:

- A ***supporting_template_method_decl*** must not be specified
- A ***union_type_decl*** must not be specified
- A ***using_external_template_statement*** must not be specified
- The **boundQuery** of the TEMPLATE whose **specification** is this ***template_specification*** must not be set
- The **effectiveQuery** of the TEMPLATE whose **specification** is this ***template_specification*** must not be set

If ***main_template_method_decl*** is a ***main_entity_type_template_method_decl***, then:

- The **boundQuery** of the TEMPLATE whose **specification** is this ***template_specification*** must be set
- An ***external_template_type_reference*** must match the **name** of a TEMPLATE
- An ***external_template_type_reference*** must not match the **name** of the TEMPLATE whose **specification** is this ***template_specification***
- If an external_template_type_reference is specified, then:

— Let THIS_TEMPLATE_NAME = the **name** of the TEMPLATE whose **specification** is this ***template_specification***

— Let SS_0 = the **specification**s (n.b. which each are ***template_specification***s) of each and every TEMPLATE whose **name** is the same as an ***external_template_type_reference*** in this ***template_specification***

— A **specification** (i.e., ***template_specification***) in SS_0 must not have an ***external_template_type_reference*** that matches by name THIS_TEMPLATE_NAME

— Let n = 0

- Recursively, if an ***external_template_type_reference*** is specified in a **specification** in SS_n, then:

  — Let SS_{n+1} = the **specification**s of each and every TEMPLATE whose **name** is the same as an ***external_template_type_reference*** in SS_n

  — A **specification** (i.e., ***template_specification***) in SS_{n+1} must not have an ***external_template_type_reference*** that matches by name THIS_TEMPLATE_NAME

- The ***external_template_type_reference*** of two or more ***using_external_template_statement***s must not be the same

- The ***template_element_type_name*** of a ***supporting_template_method_decl*** or ***union_type_decl***:

  — Must not be the same as any ***using_external_template_statement***'s ***external_template_type_reference***

  — Must not be the same as the **name** of the TEMPLATE whose **specification** is this ***template_specification***

  — Must not be the same as the ***template_element_type_name*** of any other ***supporting_template_method_decl*** or ***union_type_decl***

  — Must not be the same as the **name** of a **type** of any CHARACTERISTIC referenced by a ***designated_entity_characteristic_reference_statement***

- For a given ***supporting_template_method_decl*** or ***union_type_decl***:

  — Given a sequence of ***entity_type_structured_template_element_type_references*** from a ***main_entity_type_template_method_decl*** to that ***supporting_template_method_decl*** or ***union_type_decl***:

    - Let CP = a normalized, canonical ***designated_entity_type_reference_path*** from the ***query_selected_entity_type_reference_or_alias*** of the ***main_entity_type_template_method_decl***'s ***primary_entity_type_template_method_parameter*** to the ***query_selected_entity_type_reference_or_alias*** of that ***supporting_template_method_decl***'s or ***union_type_decl***'s ***entity_type_structured_template_element_declared_parameter_expression*** constructed by combining head-to-tail the ***designated_entity_type_reference_path*** associated with each ***entity_type_structured_template_element_type_reference*** in sequence and normalized by removing any adjacent ***query_selected_entity_type_reference_or_aliases*** that match either the ***SELECTED_ENTITY_ALIAS*** or ***ENTITY_TYPE_REFERENCE*** of the same ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***

- Two or more ***query_selected_entity_type_reference_or_aliases*** in CP must not match either the ***SELECTED_ENTITY_ALIAS*** or ***ENTITY_TYPE_REFERENCE*** of the same ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***
- The CP for all sequences of ***entity_type_structured_template_element_type_references*** from a ***main_entity_type_template_method_decl*** to that ***supporting_template_method_decl*** or ***union_type_decl*** must be the same

If an ***entity_type_structured_template_element_member*** is ***structured_template_element_type_reference_statement***, then:

- If ***entity_type_structured_template_element_type_reference*** is specified, then ***entity_type_structured_template_element_type_reference*** must match by name:
  - The ***template_element_type_name*** of a ***supporting_entity_type_template_method_decl*** or a ***union_type_decl*** of this ***template_specification***, or
  - An ***external_template_type_reference*** that is the **name** of the TEMPLATE whose ***main_template_method_decl*** is ***main_entity_type_template_method_decl***
- If ***equivalent_entity_type_template_method_reference*** is specified, then ***equivalent_entity_type_template_method_reference*** must match by name:
  - The ***template_element_type_name*** of a ***supporting_equivalent_entity_type_template_method_decl*** of this ***template_specification***, or
  - An ***external_template_type_reference*** that is the **name** of the TEMPLATE whose ***main_template_method_decl*** is ***main_equivalent_entity_type_template_method_decl***

using_external_template_statement

// No contextually relevant rules for instances of this term.

structured_template_element_type_decl

// No contextually relevant rules for instances of this term.

main_template_method_decl

// No contextually relevant rules for instances of this term.

main_entity_type_template_method_decl

The ***query_selected_entity_type_reference_or_alias*** of two or more ***entity_type_structured_template_element_declared_parameter_expression***s must not match by name either the ***SELECTED_ENTITY_ALIAS*** or ***ENTITY_TYPE_REFERENCE*** of the same ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***.

All ***entity_type_structured_template_element_declared_parameter_alias***es must be unique.

A ***primary_entity_type_template_method_parameter*** must specified.

kw_varargs must not be specified.

If ***entity_type_structured_template_element_member*** is ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***:

- If ***designated_entity_non_entity_type_characteristic_reference*** is ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:
  - — One of the conditions below must be satisfied, resolving it to the CHARACTERISTIC identified by the first rule satisfied in priority order:
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of a CHARACTERISTIC that is in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT*** and is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches by name a ***PROJECTED_CHARACTERISTIC_ALIAS*** of an ***EXPLICIT_SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of one and only one CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***

    If the CHARACTERISTIC (determined above) is not a **composition** of the ***SELECTED_ENTITY*** referenced by ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** composing that CHARACTERISTIC.
- Otherwise, ***designated_entity_non_entity_type_characteristic_reference*** is not ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:
  - — If the first ***query_selected_entity_type_reference_or_alias*** in ***designated_entity_type_reference_path*** does not match by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***'s ***designated_entity_type_reference_path***
- If ***idlstruct_member_reference*** is specified, then the **type** of the CHARACTERISTIC referenced by ***explicit_designated_entity_non_entity_type_characteristic_reference_expression*** must be an

IDLSTRUCT, and ***idlstruct_member_reference*** must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

If ***entity_type_structured_template_element_member*** is ***designated_entity_non_entity_type_characteristic_wildcard_reference***:

- If ***designated_entity_type_reference_path*** is specified:
  - If the first ***query_selected_entity_type_reference_or_alias*** in ***explicit_entity_type_reference_join_path*** matches by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then:
    - There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
  - Otherwise, the first ***query_selected_entity_type_reference_or_alias*** in ***explicit_entity_type_reference_join_path*** does not match by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then:
    - There must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***designated_entity_non_entity_type_characteristic_wildcard_reference***'s ***designated_entity_type_reference_path***
    - There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the fully inferred ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***.
- Otherwise, ***designated_entity_type_reference_path*** is not specified, then:
  - There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***

If ***entity_type_structured_template_element_member*** is ***structured_template_element_type_reference_statement***, then:

- If ***entity_type_structured_template_element_type_reference*** is specified, then:
  - An ***entity_type_structured_template_element_declared_parameter_reference*** must not be specified

— An ***optional_structured_template_element_type_reference_parameter*** must not be specified

- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***supporting_entity_type_template_method_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***supporting_template_method_decl***'s ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
- If ***inline_annotation*** is specified, then:
  - For each adjacent pair of ***query_selected_entity_type_reference_or_aliases*** in the ***primary_structured_template_element_type_reference_parameter***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path***:
    - Let L = the ***SELECTED_ENTITY*** referenced by the left ***query_selected_entity_type_reference_or_alias*** of the pair
    - Let R = the ***SELECTED_ENTITY*** referenced by the right (i.e., immediately following) ***query_selected_entity_type_reference_or_alias*** of the pair
    - Let J = the *"Join_Characteristic"* for L and R
  - If J is a **composition** or **participant** of L, then:
    - The upper bound is the **upperBound** of J
    - If R is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***, then the lower bound is 0
    - Otherwise, R is not a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***; in this case, the lower bound is the lowerBound of J
  - If J is a **composition** R, then:
    - The upper bound is 1
    - If L is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***, then the lower bound is 0
    - Otherwise, L is not a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***; in this case, the lower bound is 1
  - If J is a **participant** of R, then:
    - The upper bound is the **sourceUpperBound** of J

- If L is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***, then the lower bound for J is 0
- Otherwise, L is not a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***; in this case, the lower bound is the sourceLowerBound of J

Using the definitions above:

- The lower bound and the upper bound for all Js in the ***primary_structured_template_element_type_reference_parameter***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path*** must be 1

- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***union_type_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***union_type_decl***'s ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - ***inline_annotation*** must not be specified
- If ***entity_type_structured_template_element_type_reference*** matches by name an ***external_template_type_reference***, then:
  - ***inline_annotation*** must not be specified
  - Let T = the TEMPLATE whose **name** is ***external_template_type_reference***
  - Let M = ***main_entity_type_template_method_decl*** in T
  - Let Q = the **specification** of the QUERY that is the **boundQuery** of T
  - Let RE = the ***SELECTED_ENTITY*** referenced by the ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
  - Let DE = the ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** of Q referenced by M's ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - RE's ***ENTITY_TYPE_REFERENCE*** must match by name DE's ***ENTITY_TYPE_REFERENCE***
- If ***equivalent_entity_type_template_method_reference*** is specified, then:
  - ***inline_annotation*** must be specified
  - Let M = an ***equivalent_entity_type_template_method_decl*** where:

        - If ***equivalent_entity_type_template_method_reference*** matches by name an ***external_template_type_reference***, then M is the ***equivalent_entity_type_template_method_decl*** of the ***main_equivalent_entity_type_template_method_decl*** of the TEMPLATE whose **name** is ***external_template_type_reference***
        - Otherwise, M is the ***equivalent_entity_type_template_method_decl*** of the ***supporting_equivalent_entity_type_template_method_decl*** whose ***template_element_type_name*** is ***equivalent_entity_type_template_method_reference***

- If ***entity_type_structured_template_element_declared_parameter_reference*** is specified for the first ***structured_template_element_type_reference_parameter*** in ***structured_template_element_type_reference_parameter_list***, then:
    - An ***entity_type_structured_template_element_declared_parameter_reference*** must be specified for all ***structured_template_element_type_reference_parameter***s in a ***structured_template_element_type_reference_parameter_list***
    - All ***entity_type_structured_template_element_declared_parameter_reference***s must be unique
    - There must be one ***structured_template_element_type_reference_parameter*** in ***structured_template_element_type_reference_parameter_list*** for each ***equivalent_entity_type_template_method_parameter*** in ***equivalent_entity_type_template_method_parameter_list*** of M
    - For each ***structured_template_element_type_reference_parameter*** in ***structured_template_element_type_reference_parameter_list***:
        - The ***structured_template_element_type_reference_parameter*** 's ***entity_type_structured_template_element_declared_parameter_reference*** must match by name an ***equivalent_entity_type_template_method_parameter*** of M
    - For each ***equivalent_entity_type_template_method_member*** of M whose ***equivalent_entity_type_template_method_parameter_reference*** matches by name the ***structured_template_element_type_reference_parameter*** 's ***entity_type_structured_template_element_declared_parameter_reference***:
        - There must a **composition** in the ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** whose **rolename** matches by name the ***equivalent_entity_type_template_method_member***'s ***equivalent_entity_type_template_method_characteristic_reference***
        - If ***idlstruct_member_reference*** is specified, then the **type** of the **composition** in the ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be an IDLSTRUCT, and ***idlstruct_member_reference*** must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

Otherwise, an ***entity_type_structured_template_element_declared_parameter_reference*** must not be specified for any ***structured_template_element_type_reference_parameter***s in a ***structured_template_element_type_reference_parameter_list***, then:

- There must be one ***structured_template_element_type_reference_parameter*** in ***structured_template_element_type_reference_parameter_list*** for each ***equivalent_entity_type_template_method_parameter*** in ***equivalent_entity_type_template_method_parameter_list*** of M
- For each ***structured_template_element_type_reference_parameter*** in ***structured_template_element_type_reference_parameter_list***:
  - Let RP = the current ***structured_template_element_type_reference_parameter***
  - Let EP = the ***equivalent_entity_type_template_method_parameter_reference*** whose position in the ***equivalent_entity_type_template_method_parameter_list*** of M is the same as RP in ***structured_template_element_type_reference_parameter_list***
- For each ***equivalent_entity_type_template_method_member*** of M whose ***equivalent_entity_type_template_method_parameter_reference*** matches by name EP:
  - There must a **composition** in the ***SELECTED_ENTITY*** referenced by RP's ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** whose **rolename** matches by name the ***equivalent_entity_type_template_method_member***'s ***equivalent_entity_type_template_method_characteristic_reference***
  - If ***idlstruct_member_reference*** is specified, then the **type** of the **composition** in the ***SELECTED_ENTITY*** referenced by RP's ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be an IDLSTRUCT, and ***idlstruct_member_reference*** must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

primary_entity_type_template_method_parameter

// No contextually relevant rules for instances of this term.

optional_entity_type_template_method_parameter_list

// No contextually relevant rules for instances of this term.

entity_type_template_method_parameter

// No contextually relevant rules for instances of this term.

main_equivalent_entity_type_template_method_decl

// No contextually relevant rules for instances of this term.

supporting_template_method_decl

// No contextually relevant rules for instances of this term.

supporting_entity_type_template_method_decl

If ***entity_type_structured_template_element_member*** is ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***:

- If ***designated_entity_non_entity_type_characteristic_reference*** is ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:
  - — One of the conditions below must be satisfied, resolving it to the CHARACTERISTIC identified by the first rule satisfied in priority order:
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of a CHARACTERISTIC that is in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT*** and is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches by name a ***PROJECTED_CHARACTERISTIC_ALIAS*** of an ***EXPLICIT_SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of one and only one CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***

    If the CHARACTERISTIC (determined above) is not a **composition** of the ***SELECTED_ENTITY*** referenced by ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** composing that CHARACTERISTIC.
- Otherwise, ***designated_entity_non_entity_type_characteristic_reference*** is not ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:
  - — If the first ***query_selected_entity_type_reference_or_alias*** in ***designated_entity_type_reference_path*** does not match by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***'s ***designated_entity_type_reference_path***
- If ***idlstruct_member_reference*** is specified, then the **type** of the CHARACTERISTIC referenced by ***explicit_designated_entity_non_entity_type_characteristic_reference_expression*** must be an

IDLSTRUCT, and ***idlstruct_member_reference*** must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

If ***entity_type_structured_template_element_member*** is ***designated_entity_non_entity_type_characteristic_wildcard_reference***:

- If ***designated_entity_type_reference_path*** is specified:
  - — If the first ***query_selected_entity_type_reference_or_alias*** in ***explicit_entity_type_reference_join_path*** matches by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then:
    - There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
  - — Otherwise, the first ***query_selected_entity_type_reference_or_alias*** in ***explicit_entity_type_reference_join_path*** does not match by name the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then:
    - There must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***designated_entity_non_entity_type_characteristic_wildcard_reference***'s ***designated_entity_type_reference_path***
    - There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the fully inferred ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
- Otherwise, ***designated_entity_type_reference_path*** is not specified, then:
  - — There must at least one CHARACTERISTIC in the ***PROJECTED_CHARACTER_LIST*** of the outermost ***QUERY_STATEMENT*** that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***

If ***entity_type_structured_template_element_member*** is ***structured_template_element_type_reference_statement***, then:

- If ***entity_type_structured_template_element_type_reference*** is specified, then:
  - — An ***entity_type_structured_template_element_declared_parameter_reference*** must not be specified

— An ***optional_structured_template_element_type_reference_parameter*** must not be specified

- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***supporting_entity_type_template_method_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***supporting_template_method_decl***'s ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - If ***inline_annotation*** is specified, then:
    - Let JS = the cumulative set of all *"Join_Characteristic"*s (as defined in the rules for term ***designated_entity_type_reference_path***) in the canonical ***designated_entity_type_reference_path*** from the ***query_selected_entity_type_reference_or_alias*** of the ***main_entity_type_template_method_decl***'s ***primary_entity_type_template_method_parameter*** to the ***query_selected_entity_type_reference_or_alias*** of this ***supporting_template_method_decl***'s ***entity_type_structured_template_element_declared_parameter_expression*** (constructed by combining head-to-tail the ***designated_entity_type_reference_path*** associated with each ***entity_type_structured_template_element_type_reference*** in sequence)
  - For each adjacent pair of ***query_selected_entity_type_reference_or_aliases*** in the ***primary_structured_template_element_type_reference_parameter***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path***:
    - Let L = the ***SELECTED_ENTITY*** referenced by the left ***query_selected_entity_type_reference_or_alias*** of the pair
    - Let R = the ***SELECTED_ENTITY*** referenced by the right (i.e., immediately following) ***query_selected_entity_type_reference_or_alias*** of the pair
    - Let J = the *"Join_Characteristic"* for L and R
  - If J is a **composition** or **participant** of L, then:
    - If J is in JS, then both the lower bound and upper bound for J is 1
    - If J is not in JS, then the lower bound is the **lowerBound** of J and upper bound is the **upperBound** of J
    - If R is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***, then the lower bound for J is 0 (if true, this supersedes any previously determined lower bound for J)
  - If J is a **composition** or **participant** of R, then:
    - If J is in JS, then the lower bound and upper bound for J is 1
    - If J is not in JS, then:

- If J is a **composition** of R, then the lower bound and upper bound for J is 1
- If J is a **participant** of R, then the lower bound is the **sourceLowerBound** of J and upper bound is the **sourceUpperBound** of J
- If L is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE CHARARACTERISTIC_BASIS***, then the lower bound for J is 0 (if true, this supersedes any previously determined lower bound for J)

Using the definitions above:

- The lower bound and the upper bound for all Js in the ***primary_structured_template_element_type_reference_parameter***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path*** must be 1

- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***union_type_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***union_type_decl***'s ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - ***inline_annotation*** must not be specified
- If ***entity_type_structured_template_element_type_reference*** matches by name an ***external_template_type_reference***, then:
  - ***inline_annotation*** must not be specified
  - Let T = the TEMPLATE whose **name** is ***external_template_type_reference***
  - Let M = ***main_entity_type_template_method_decl*** in T
  - Let Q = the **specification** of the QUERY that is the **boundQuery** of T
  - Let RE = the ***SELECTED_ENTITY*** referenced by the ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
  - Let DE = the ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** of Q referenced by M's ***primary_entity_type_template_method_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - RE's ***ENTITY_TYPE_REFERENCE*** must match by name DE's ***ENTITY_TYPE_REFERENCE***
- If ***equivalent_entity_type_template_method_reference*** is specified, then:
  - ***inline_annotation*** must be specified
  - Let M = an ***equivalent_entity_type_template_method_decl*** where:
    - If ***equivalent_entity_type_template_method_reference*** matches by name an ***external_template_type_reference***, then M is the

equivalent_entity_type_template_method_decl of the *main_equivalent_entity_type_template_method_decl* of the TEMPLATE whose **name** is *external_template_type_reference*

— Otherwise, M is the *equivalent_entity_type_template_method_decl* of the *supporting_equivalent_entity_type_template_method_decl* whose *template_element_type_name* is *equivalent_entity_type_template_method_reference*

If *entity_type_structured_template_element_declared_parameter_reference* is specified for the first *structured_template_element_type_reference_parameter* in *structured_template_element_type_reference_parameter_list*, then:

- An *entity_type_structured_template_element_declared_parameter_reference* must be specified for all *structured_template_element_type_reference_parameter*s in a *structured_template_element_type_reference_parameter_list*
- All *entity_type_structured_template_element_declared_parameter_reference*s must be unique
- There must be one *structured_template_element_type_reference_parameter* in *structured_template_element_type_reference_parameter_list* for each *equivalent_entity_type_template_method_parameter* in *equivalent_entity_type_template_method_parameter_list* of M
- For each *structured_template_element_type_reference_parameter* in *structured_template_element_type_reference_parameter_list*:
  - The *structured_template_element_type_reference_parameter* 's *entity_type_structured_template_element_declared_parameter_reference* must match by name an *equivalent_entity_type_template_method_parameter* of M
  - For each *equivalent_entity_type_template_method_member* of M whose *equivalent_entity_type_template_method_parameter_reference* matches by name the *structured_template_element_type_reference_parameter* 's *entity_type_structured_template_element_declared_parameter_reference*:
    - There must a **composition** in the *SELECTED_ENTITY* referenced by the *structured_template_element_type_reference_parameter*'s *designated_entity_type_reference*'s *query_selected_entity_type_reference_or_alias* whose **rolename** matches by name the *equivalent_entity_type_template_method_member*'s *equivalent_entity_type_template_method_characteristic_reference*
    - If *idlstruct_member_reference* is specified, then the **type** of the **composition** in the *SELECTED_ENTITY* referenced by the *structured_template_element_type_reference_parameter*'s *designated_entity_type_reference*'s *query_selected_entity_type_reference_or_alias* must be an IDLSTRUCT, and *idlstruct_member_reference* must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

Otherwise, an *entity_type_structured_template_element_declared_parameter_reference* must not be specified for any *structured_template_element_type_reference_parameter*s in a *structured_template_element_type_reference_parameter_list*, then:

- There must be one *structured_template_element_type_reference_parameter* in *structured_template_element_type_reference_parameter_list* for each

equivalent_entity_type_template_method_parameter in *equivalent_entity_type_template_method_parameter_list* of M

- For each *structured_template_element_type_reference_parameter* in *structured_template_element_type_reference_parameter_list*:
  - Let RP = the current *structured_template_element_type_reference_parameter*
  - Let EP = the *equivalent_entity_type_template_method_parameter_reference* whose position in the *equivalent_entity_type_template_method_parameter_list* of M is the same as RP in *structured_template_element_type_reference_parameter_list*
  - For each *equivalent_entity_type_template_method_member* of M whose *equivalent_entity_type_template_method_parameter_reference* matches by name EP:
    - There must a **composition** in the *SELECTED_ENTITY* referenced by RP's *designated_entity_type_reference*'s *query_selected_entity_type_reference_or_alias* whose **rolename** matches by name the *equivalent_entity_type_template_method_member*'s *equivalent_entity_type_template_method_characteristic_reference*
    - If *idlstruct_member_reference* is specified, then the **type** of the **composition** in the *SELECTED_ENTITY* referenced by RP's *designated_entity_type_reference*'s *query_selected_entity_type_reference_or_alias* must be an IDLSTRUCT, and *idlstruct_member_reference* must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

supporting_equivalent_entity_type_template_method_decl

// No contextually relevant rules for instances of this term.

entity_type_template_method_body

All *entity_type_template_method_member*s must result in a unique set of member names, where the member name(s) of a given *entity_type_template_method_member* is (are):

- If *entity_type_template_method_member* is *explicit_designated_entity_non_entity_type_characteristic_reference_expression*, then the member name is:
  - *structured_template_element_member_name*, if specified
  - Otherwise, *idlstruct_member_reference*, if specified
  - Otherwise, *designated_entity_type_reference_path* is specified, then *query_projected_non_entity_type_characteristic_reference*
  - Otherwise, *query_projected_non_entity_type_characteristic_reference_or_alias*
- If *entity_type_template_method_member* is *designated_entity_non_entity_type_characteristic_wildcard_reference*, then each CHARACTERISTIC in the *PROJECTED_CHARACTER_LIST* of the outermost *QUERY_STATEMENT* that is a composition of the *SELECTED_ENTITY* referenced by the explicitly specified or unambiguously inferred *designated_entity_type_reference_path*'s *designated_entity_type_reference's query_selected_entity_type_reference_or_alias* is effectively a member, and the member's name is the name of the CHARACTERISTIC

- If ***entity_type_template_method_member*** is ***structured_template_element_type_reference_statement***, then:
    - If ***entity_type_structured_template_element_type_reference*** is specified, then:
        - If ***inline_annotation*** is specified, then the members name(s) is (are) determined by applying this rule to the ***entity_type_template_method_body*** of the ***supporting_entity_type_template_method_decl*** or ***main_entity_type_template_method_decl*** referenced by ***entity_type_structured_template_element_type_reference***
    - Otherwise, ***inline_annotation*** is not specified, then the member's name is ***structured_template_element_member_name***
- If ***equivalent_entity_type_template_method_reference*** is specified, then each ***equivalent_entity_type_template_element_member*** of the ***supporting_equivalent_entity_type_template_method_decl*** or ***main_equivalent_entity_type_template_method_decl*** referenced by ***equivalent_entity_type_template_method_reference*** is effectively a member, and the member's name is the ***equivalent_entity_type_template_element_member***'s:
    - ***structured_template_element_member_name***, if specified
    - Otherwise, ***idlstruct_member_reference***, if specified
    - Otherwise, ***equivalent_entity_type_template_method_characteristic_reference***

entity_type_template_method_member

// No contextually relevant rules for instances of this term.

equivalent_entity_type_template_method_decl

An ***equivalent_entity_type_template_method_parameter_reference*** must match by name an ***equivalent_entity_type_template_method_parameter***.

equivalent_entity_type_template_method_parameter_list

All ***equivalent_entity_type_template_method_parameter***s must be unique.

equivalent_entity_type_template_method_body

All ***equivalent_entity_type_template_element_member***s must have a unique member name, where the member name of a given ***equivalent_entity_type_template_element_member*** is:

- ***structured_template_element_member_name***, if specified
- Otherwise, ***idlstruct_member_reference***, if specified
- Otherwise, ***equivalent_entity_type_template_method_characteristic_reference***

Two or more ***equivalent_entity_type_template_element_member***s must not have, as a set, the same ***equivalent_entity_type_template_method_parameter_reference***, ***idlstruct_member_reference***, ***equivalent_entity_type_template_method_characteristic_reference***, and ***optional_annotation***.

equivalent_entity_type_template_method_member

// No contextually relevant rules for instances of this term.

equivalent_entity_type_template_element_member_statement

// No contextually relevant rules for instances of this term.

union_type_decl

If ***discriminator_type*** is a ***designated_entity_enumeration_type_characteristic_reference***, then:

- If ***designated_entity_enumeration_type_characteristic_reference*** is ***query_projected_enumeration_type_characteristic_reference_or_alias***, then:
  - — One of the conditions below must be satisfied, resolving it to the CHARACTERISTIC identified by the first rule satisfied in priority order:
    - ***query_projected_enumeration_type_characteristic_reference_or_alias*** matches the **rolename** of a CHARACTERISTIC that is in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT*** and is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
    - ***query_projected_enumeration_type_characteristic_reference_or_alias*** matches by name a ***PROJECTED_CHARACTERISTIC_ALIAS*** of an ***EXPLICIT_SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***
    - ***query_projected_enumeration_type_characteristic_reference_or_alias*** matches the **rolename** of one and only one CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***

    If the CHARACTERISTIC (determined above) is not a **composition** of the ***SELECTED_ENTITY*** referenced by ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** composing that CHARACTERISTIC.
- Otherwise, ***designated_entity_enumeration_type_characteristic_reference*** is not ***query_projected_enumeration_type_characteristic_reference_or_alias***, then:
  - — If the first ***query_selected_entity_type_reference_or_alias*** in ***designated_entity_type_reference_path*** does not match by name the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***discriminator_type***'s ***designated_entity_type_reference_path***

Given the following definitions:

- Let JS = the cumulative set of all *"Join_Characteristic"*s (as defined in the rules for term ***designated_entity_type_reference_path***) in the canonical ***designated_entity_type_reference_path*** from the ***query_selected_entity_type_reference_or_alias*** of the ***main_entity_type_template_method_decl***'s ***primary_entity_type_template_method_parameter*** to the ***query_selected_entity_type_reference_or_alias*** of this ***union_type_decl***'s ***entity_type_structured_template_element_declared_parameter_expression*** (constructed by combining head-to-tail the ***designated_entity_type_reference_path*** associated with each ***entity_type_structured_template_element_type_reference*** in sequence)

For each adjacent pair of ***query_selected_entity_type_reference_or_alias***es in the ***discriminator_type***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path***:

- Let L = the ***SELECTED_ENTITY*** referenced by the left ***query_selected_entity_type_reference_or_alias*** of the pair
- Let R = the ***SELECTED_ENTITY*** referenced by the right (i.e., immediately following) ***query_selected_entity_type_reference_or_alias*** of the pair
- Let J = the *"Join_Characteristic"* for L and R

  If J is a **composition** or **participant** of L, then:

  — If J is in JS, then both the lower bound and upper bound for J is 1

  — If J is not in JS, then the lower bound is the **lowerBound** of J and upper bound is the **upperBound** of J

  — If R is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** ***CHARARACTERISTIC_BASIS***, then the lower bound for J is 0 (if true, this supersedes any previously determined lower bound for J)

  If J is a **composition** or **participant** of R, then:

  — If J is in JS, then the lower bound and upper bound for J is 1

  — If J is not in JS, then:

    - If J is a **composition** of R, then the lower bound and upper bound for J is 1
    - If J is a **participant** of R, then the lower bound is the **sourceLowerBound** of J and upper bound is the **sourceUpperBound** of J

  — If L is a ***SELECTED_ENTITY*** that is the ***SELECTED_ENTITY_REFERENCE*** of a ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** ***CHARARACTERISTIC_BASIS***, then the lower bound for J is 0 (if true, this supersedes any previously determined lower bound for J)

  Using the definitions above:

  — The lower bound and the upper bound for all Js in the ***discriminator_type***'s explicitly specified or unambiguously inferred ***designated_entity_type_reference_path*** must be 1, and both the **lowerBound** and **upperBound** of the CHARACTERISTIC referenced by ***designated_entity_enumeration_type_characteristic_reference*** must also be 1

- Each ***case_label_literal*** must be an ***enum_literal_reference_expression*** whose:

— ***enumeration_type_reference*** must match the **name** of the ENUMERATED in the MEASUREMENT realized by the **type** of the COMPOSITION referenced by ***designated_entity_enumeration_type_characteristic_reference***, and

— ***enumeration_literal_reference*** must match the **name** of a ENUMERATIONLABEL that is a **label** of the above ENUMERATED; if an ENUMERATIONCONSTRAINT is specified for the above MEASUREMENT, then that ENUMERATIONLABEL must also be an **allowedValue** of that ENUMERATIONCONSTRAINT

If ***entity_type_structured_template_element_member*** is ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***:

- If ***designated_entity_non_entity_type_characteristic_reference*** is ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:

  — One of the conditions below must be satisfied, resolving it to the CHARACTERISTIC identified by the first rule satisfied in priority order:

    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of a CHARACTERISTIC that is in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT*** and is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches by name a ***PROJECTED_CHARACTERISTIC_ALIAS*** of an ***EXPLICIT_SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***
    - ***query_projected_non_entity_type_characteristic_reference_or_alias*** matches the **rolename** of one and only one CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***

  If the CHARACTERISTIC (determined above) is not a **composition** of the ***SELECTED_ENTITY*** referenced by ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** composing that CHARACTERISTIC.

- Otherwise, ***designated_entity_non_entity_type_characteristic_reference*** is not ***query_projected_non_entity_type_characteristic_reference_or_alias***, then:

  — If the first ***query_selected_entity_type_reference_or_alias*** in ***designated_entity_type_reference_path*** does not match by name the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** or, if specified, its ***entity_type_structured_template_element_declared_parameter_alias***, then there must be an unambiguous ***designated_entity_type_reference_path*** (inferred from the ***ENTITY_EXPRESSION*** of the outermost ***QUERY_STATEMENT***) from the ***SELECTED_ENTITY*** referenced by the ***union_parameter***'s ***query_selected_entity_type_reference_or_alias*** to the ***SELECTED_ENTITY*** referenced by the first ***query_selected_entity_type_reference_or_alias*** in the ***explicit_designated_entity_non_entity_type_characteristic_reference_expression***'s ***designated_entity_type_reference_path***

- If ***idlstruct_member_reference*** is specified, then the **type** of the CHARACTERISTIC referenced by ***explicit_designated_entity_non_entity_type_characteristic_reference_expression*** must be an IDLSTRUCT, and ***idlstruct_member_reference*** must match the **rolename** of an IDLCOMPOSITION that is a **composition** of that IDLSTRUCT

If ***entity_type_structured_template_element_member*** is ***structured_template_element_type_reference_statement***, then:

- If ***entity_type_structured_template_element_type_reference*** is specified, then:
  - An ***entity_type_structured_template_element_declared_parameter_reference*** must not be specified
  - An ***optional_structured_template_element_type_reference_parameter*** must not be specified
- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***supporting_entity_type_template_method_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***supporting_template_method_decl***'s ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
- If ***entity_type_structured_template_element_type_reference*** matches by name the ***template_element_type_name*** of a ***union_type_decl***, then:
  - The ***SELECTED_ENTITY*** referenced by the ***structured_template_element_type_reference_expression***'s ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference_path***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** must be the same as the ***SELECTED_ENTITY*** referenced by the ***union_type_decl***'s ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
- If ***entity_type_structured_template_element_type_reference*** matches by name an ***external_template_type_reference***, then:
  - Let T = the TEMPLATE whose **name** is ***external_template_type_reference***
  - Let M = ***main_entity_type_template_method_decl*** in T
  - Let Q = the **specification** of the QUERY that is the **boundQuery** of T
  - Let RE = the ***SELECTED_ENTITY*** referenced by the ***primary_structured_template_element_type_reference_parameter***'s ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***
  - Let DE = the ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** of Q referenced by M's ***union_parameter***'s ***query_selected_entity_type_reference_or_alias***
  - RE's ***ENTITY_TYPE_REFERENCE*** must match by name DE's ***ENTITY_TYPE_REFERENCE***

- An ***equivalent_entity_type_template_method_reference*** must not be specified

union_parameter

// No contextually relevant rules for instances of this term.

union_body

// No contextually relevant rules for instances of this term.

union_switch_statement

If ***discriminator_type*** is ***idlunsigned_short***, then each ***case_label_literal*** must be an ***idldiscriminator_type_literal*** whose value is an integer in the range 0 … 2^{16}-1.

If ***discriminator_type*** is ***idlunsigned_long***, then each ***case_label_literal*** must be an ***idldiscriminator_type_literal*** value is an integer in the range 0 … 2^{32}-1.

If ***discriminator_type*** is ***idlunsigned_long_long***, then each ***case_label_literal*** must be an ***idldiscriminator_type_literal*** value is an integer in the range 0 … 2^{64}-1.

discriminator_type

// No contextually relevant rules for instances of this term.

union_switch_body

The ***kw_default case_label*** may be specified at most once.

There must be at least ***case_label*** with a ***case_label_literal***.

All ***case_label_literals*** must be unique.

case_expression

// No contextually relevant rules for instances of this term.

case_label

// No contextually relevant rules for instances of this term.

case_label_literal

// No contextually relevant rules for instances of this term.

union_member

A ***designated_entity_non_entity_type_characteristic_wildcard_reference*** must not be specified.

A ***structured_template_element_type_reference_statement*** must not specify an ***inline_annotation***.

entity_type_structured_template_element_member

// No contextually relevant rules for instances of this term.

entity_type_structured_template_element_member_statement

// No contextually relevant rules for instances of this term.

optional_annotation

// No contextually relevant rules for instances of this term.

inline_annotation

// No contextually relevant rules for instances of this term.

designated_entity_characteristic_reference_statement

// No contextually relevant rules for instances of this term.

explicit_designated_entity_non_entity_type_characteristic_reference_expression

// No contextually relevant rules for instances of this term.

structured_template_element_type_reference_statement

// No contextually relevant rules for instances of this term.

structured_template_element_type_reference_expression

// No contextually relevant rules for instances of this term.

structured_template_element_type_reference_parameter_list

// No contextually relevant rules for instances of this term.

primary_structured_template_element_type_reference_parameter

// No contextually relevant rules for instances of this term.

optional_structured_template_element_type_reference_parameter

// No contextually relevant rules for instances of this term.

structured_template_element_type_reference_parameter

// No contextually relevant rules for instances of this term.

external_template_type_reference

An ***external_template_type_reference*** must match the **name** a TEMPLATE that is an **element** of a PLATFORMDATAMODEL.

entity_type_structured_template_element_type_reference

// No contextually relevant rules for instances of this term.

entity_type_structured_template_element_declared_parameter_reference

// No contextually relevant rules for instances of this term.

entity_type_structured_template_element_declared_parameter_expression

A ***query_selected_entity_type_reference_or_alias*** must match by name either the ***SELECTED_ENTITY_ALIAS*** or the ***ENTITY_TYPE_REFERENCE*** of one and only one ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***.

entity_type_structured_template_element_declared_parameter_alias

An ***entity_type_structured_template_element_declared_parameter_alias*** must not be the same as the ***SELECTED_ENTITY_ALIAS*** or ***ENTITY_TYPE_REFERENCE*** of any ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***.

An ***entity_type_structured_template_element_declared_parameter_alias*** must not be the same as any ***PROJECTED_CHARACTERISTIC_ALIAS*** in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***.

An ***entity_type_structured_template_element_declared_parameter_alias*** must not be a reserved word as defined by OCL constraint face::Element::isReservedWord specified in the FACE Technical Standard §J.6.1: OCL Constraint Helper Methods.

structured_template_element_member_name

A ***structured_template_element_member_name*** must not be a reserved word as defined by OCL constraint face::Element::isReservedWord specified in the FACE Technical Standard §J.6.1: OCL Constraint Helper Methods.

template_element_type_name

A ***template_element_type_name*** must not be a reserved word as defined by OCL constraint face::Element::isReservedWord specified in the FACE Technical Standard §J.6.1: OCL Constraint Helper Methods.

A ***template_element_type_name*** must not be "*main*".

equivalent_entity_type_template_method_reference

// No contextually relevant rules for instances of this term.

equivalent_entity_type_template_method_parameter

// No contextually relevant rules for instances of this term.

designated_equivalent_entity_non_entity_type_characteristic_reference

// No contextually relevant rules for instances of this term.

equivalent_entity_type_template_method_parameter_reference

// No contextually relevant rules for instances of this term.

equivalent_entity_type_template_method_characteristic_reference

// No contextually relevant rules for instances of this term.

designated_entity_non_entity_type_characteristic_reference

If ***designated_entity_type_reference_path*** is specified, then ***query_projected_non_entity_type_characteristic_reference*** must match the **rolename** of a CHARACTERISTIC that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***.

designated_entity_non_entity_type_characteristic_wildcard_reference

// No contextually relevant rules for instances of this term.

designated_entity_enumeration_type_characteristic_reference

If ***designated_entity_type_reference_path*** is specified, then ***query_projected_enumeration_type_characteristic_reference*** must match the **rolename** of a CHARACTERISTIC that is a **composition** of the ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***.

designated_entity_type_reference_path

Two or more ***query_selected_entity_type_reference_or_alias***es must not match either the ***SELECTED_ENTITY_ALIAS*** or ***ENTITY_TYPE_REFERENCE*** of the same ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***.

If an ***explicit_entity_type_reference_join_path*** is specified, then:

- Let N = the number of ***query_selected_entity_type_reference_or_alias***es
- If N > 1, then for each ***query_selected_entity_type_reference_or_alias*** [n] where n < N, there must a ***ENTITY_TYPE_CHARACTERISTIC_EQUIVALENCE_EXPRESSION*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** whose ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** is a reference to a CHARACTERISTIC that specifically joins ***query_selected_entity_type_reference_or_alias*** [n] and ***query_selected_entity_type_reference_or_alias*** [n + 1], where the CHARACTERISTIC is:

— A **composition** or **participant** of the ***SELECTED_ENTITY*** referenced by ***query_selected_entity_type_reference_or_alias*** [n], and whose **type** is the ***SELECTED_ENTITY*** referenced by the immediately following ***query_selected_entity_type_reference_or_alias*** [n + 1] in ***explicit_entity_type_reference_join_path***, or

— A **composition** or **participant** of the ***SELECTED_ENTITY*** referenced by the immediately following ***query_selected_entity_type_reference_or_alias*** [n + 1] in ***explicit_entity_type_reference_join_path***, and whose **type** is the ***SELECTED_ENTITY*** referenced by that ***query_selected_entity_type_reference_or_alias*** [n]

- For the last or only ***query_selected_entity_type_reference_or_alias*** (i.e., n = N) , there must be a ***ENTITY_TYPE_CHARACTERISTIC_EQUIVALENCE_EXPRESSION*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** whose ***SELECTED_ENTITY_CHARACTERISTIC_REFERENCE*** is a reference to a CHARACTERISTIC that specifically joins ***query_selected_entity_type_reference_or_alias*** [n] and the ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***, where the CHARACTERISTIC is a **composition** or **participant** of:

  — The ***SELECTED_ENTITY*** referenced by ***query_selected_entity_type_reference_or_alias*** [n] and whose **type** is the ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias***, or

  — The ***SELECTED_ENTITY*** referenced by the ***designated_entity_type_reference***'s ***query_selected_entity_type_reference_or_alias*** and whose **type** is the ***SELECTED_ENTITY*** referenced by that ***query_selected_entity_type_reference_or_alias*** [n]

Note: In both cases above, the CHARACTERISTIC that joins a given adjacent pair of ***query_selected_entity_type_reference_or_alias***es – that is, a ***query_selected_entity_type_reference_or_alias*** and its immediately following ***query_selected_entity_type_reference_or_alias*** (e.g., [n] and [n + 1]) in ***designated_entity_type_reference_path*** – is referred to as a *"Join_Characteristic"*.

explicit_entity_type_reference_join_path

// No contextually relevant rules for instances of this term.

join_path_entity_type_reference

// No contextually relevant rules for instances of this term.

designated_entity_type_reference

// No contextually relevant rules for instances of this term.

qualified_entity_type_reference

// No contextually relevant rules for instances of this term.

entity_type_reference

// No contextually relevant rules for instances of this term.

entity_characteristic_value_qualifier

// No contextually relevant rules for instances of this term.

idlstruct_member_reference

// No contextually relevant rules for instances of this term.

enum_literal_reference_expression

// No contextually relevant rules for instances of this term.

enumeration_type_reference

// No contextually relevant rules for instances of this term.

enumeration_literal_reference

// No contextually relevant rules for instances of this term.

query_where_clause_criteria

// No contextually relevant rules for instances of this term.

query_projected_non_entity_type_characteristic_reference_or_alias

A ***query_projected_non_entity_type_characteristic_reference_or_alias*** must be a CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***.

A ***query_projected_non_entity_type_characteristic_reference_or_alias*** must be a COMPOSITION whose **type** is not an ENTITY.

query_projected_non_entity_type_characteristic_reference

A ***query_projected_non_entity_type_characteristic_reference*** must be a CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***.

A ***query_projected_non_entity_type_characteristic_reference*** must be a COMPOSITION whose **type** is not an ENTITY.

query_projected_enumeration_type_characteristic_reference_or_alias

A ***query_projected_enumeration_type_characteristic_reference_or_alias*** must be a CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***.

A ***query_projected_enumeration_type_characteristic_reference_or_alias*** must be a COMPOSITION whose **type** is an ENUMERATION that realizes a MEASUREMENT whose **measurementSystem** is the MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem*".

query_projected_enumeration_type_characteristic_reference

A ***query_projected_enumeration_type_characteristic_reference*** must be a CHARACTERISTIC in the ***PROJECTED_CHARACTERISTIC_LIST*** of the outermost ***QUERY_STATEMENT***.

A ***query_projected_enumeration_type_characteristic_reference*** must be a COMPOSITION whose **type** is an ENUMERATION that realizes a MEASUREMENT whose **measurementSystem** is the MEASUREMENTSYSTEM whose **name** is "*AbstractDiscreteSetMeasurementSystem"*.

query_selected_entity_type_reference_or_alias

A ***query_selected_entity_type_reference_or_alias*** must match by name either the ***SELECTED_ENTITY_ALIAS*** or the ***ENTITY_TYPE_REFERENCE*** of one and only one ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT***.

A ***query_selected_entity_type_reference_or_alias*** must not match the ***ENTITY_TYPE_REFERENCE*** of any ***SELECTED_ENTITY*** in the ***FROM_CLAUSE*** of the outermost ***QUERY_STATEMENT*** whose ***ENTITY_TYPE_REFERENCE*** is not unique to one ***SELECTED_ENTITY***.

idldiscriminator_type

// No contextually relevant rules for instances of this term.

idlunsigned_int

// No contextually relevant rules for instances of this term.

idlunsigned_short

// No contextually relevant rules for instances of this term.

idlunsigned_long

// No contextually relevant rules for instances of this term.

idlunsigned_long_long

// No contextually relevant rules for instances of this term.

idlboolean

// No contextually relevant rules for instances of this term.

idldiscriminator_type_literal

// No contextually relevant rules for instances of this term.

idlinteger_literal

// No contextually relevant rules for instances of this term.

idloctal_literal

// No contextually relevant rules for instances of this term.

idlhex_literal

// No contextually relevant rules for instances of this term.

idlboolean_literal

// No contextually relevant rules for instances of this term.

IDENTIFIER

An ***IDENTIFIER*** is a valid String as defined by OCL constraint face::Element::isValidIdentifier specified in the FACE Technical Standard §J.6.1: OCL Constraint Helper Methods.

D FACE Technical Standard, Edition 3.1 Data Types

A review of FACE Technical Standard, Edition 3.1 data types suggests that the following data types are the most likely candidates to prevent public release:

- datamodel.logical.Landmark
- datamodel.logical.ReferencePoint
- datamodel.logical.ReferencePointPart
- datamodel.logical.MeasurementSystem
- datamodel.logical.MeasurementSystemAxis
- datamodel.logical.CoordinateSystem
- datamodel.logical.CoordinateSystemAxis

D.1 Basis Elements

Starting with the FACE Technical Standard, Edition 3.1, the Data Architecture extends the Open Universal Domain Description Language (Open UDDL) Standard, Edition 1.0.[3] The following elements are the Basis Elements for FACE Edition 3.1, and may be defined in either the FACE Technical Standard, Edition 3.1 or the Open UDDL Standard, Edition 1.0:

- datamodel.conceptual.Observable
- datamodel.conceptual.Domain
- datamodel.conceptual.BasisEntity
- datamodel.logical.Unit
- datamodel.logical.Landmark
- datamodel.logical.ReferencePoint
- datamodel.logical.ReferencePointPart
- datamodel.logical.StandardMeasurementSystem
- datamodel.logical.MeasurementSystem
- datamodel.logical.MeasurementSystemAxis
- datamodel.logical.CoordinateSystem
- datamodel.logical.CoordinateSystemAxis
- datamodel.logical.MeasurementSystemConversion
- datamodel.logical.Boolean
- datamodel.logical.Character

[3] Open Universal Domain Description Language (Open UDDL), Edition 1.0, The Open Group Standard (C198), published by The Open Group, July 2019; refer to: www.opengroup.org/library/c198.

- datamodel.logical.Numeric
- datamodel.logical.Integer
- datamodel.logical.Natural
- datamodel.logical.NonNegativeReal
- datamodel.logical.Real
- datamodel.logical.String

D.2 Query Rules

Query rules are defined in the Open UDDL Standard, Edition 1.0.

D.3 Template Rules

Template rules are defined in the FACE Technical Standard, Edition 3.1.

Acronyms

CCB	Configuration Control Board
CR	Change Request
DIOG	Domain Interoperability Working Group
DSDM	Domain-Specific Data Model
FACE	Future Airborne Capability Environment
ID	Identifier
ITAR	International Traffic in Arms Regulations
OCL	Object Constraint Language
PCS	Portable Component Segment
PSSS	Platform-Specific Services Segment
SDM	Shared Data Model
TSS	Transport Services Segment
TWG	Technical Working Group
UoC	Unit of Conformance
UoP	Unit of Portability
USM	Unit of Portability Supplied Model
UDDL	Universal Domain Description Language