ELF 5006:2025

EXPRESS mapping language specification

Thomas Thurman
Ronald Tse

October 19, 2025
5006:2025, Version 1.0

CONTENTS

FOREWORD

The EXPRESS Language Foundation (“ELF”) is a registered public charity in the US that facilitates the education, standardization, research, promotion, definition, and usage of information modelling and programming languages, with a focus on the EXPRESS language family.

ELF works with international partners and experts across the globe, reflecting the international nature of its mission. More information about ELF is available on the official website (https://www.expresslang.org).

The procedures used to develop this document and those intended for its further maintenance are described in the ELF Directives.

In particular, the different approval criteria needed for the different types of ELF documents should be noted. This document was drafted in accordance with the editorial rules of the ELF Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ELF shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be provided in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by Technical Committee *EXPRESS*.

INTRODUCTION

0.1 General

The EXPRESS mapping language is a domain-specific language designed to express reference paths between entities in a set of EXPRESS schemas.

It provides a formal syntax and semantics for specifying how entities in one schema relate to entities in another schema through a series of navigations. These paths can include subtype and supertype relationships, forward and inverse attribute navigations, and constraints on entity attributes.

The operators and syntax to express allow for these navigations to be defined precisely and unambiguously.

The mapping language serves several purposes:

Documentation	Provides a clear, machine-readable specification of how concepts of a conceptual model described in EXPRESS maps to an implementation described in another EXPRESS schema.
Validation	Enables automated validation that mapping paths are correct and traverse valid relationships in the EXPRESS schema.
Interoperability	Ensures consistent interpretation of mappings across different tools and implementations.
Maintenance	Facilitates detection of broken mappings when schemas evolve.

0.2 Historical context

In ISO 10303 (STEP), requirements of the application modules are defined using Application Resource Models (ARMs) expressed in EXPRESS.

The implementation of these requirements is defined using Module Interpreted Models (MIMs), also expressed in EXPRESS.

The EXPRESS mapping language was developed to formally express the relationships between ARM entities and MIM entities.

1. SCOPE

This document describes the EXPRESS mapping language used to express reference paths amongst a set of EXPRESS schemas.

2. NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 10303-1:2024, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles*

ISO 10303-11:2004, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*

3. TERMS AND DEFINITIONS

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org

3.1 application resource model

ARM PREFERRED

conceptual model that represents application domain concepts in a neutral, implementation-independent manner

[SOURCE:]

3.2 module interpreted model

MIM PREFERRED

implementation model that maps ARM concepts to EXPRESS constructs using entities and relationships from integrated resources

[SOURCE:]

3.3 reference path PREFERRED

sequence of navigation steps through EXPRESS schema entity relationships, expressing how an entity in one schema relates to an entity in another schema

3.4 subtype relationship PREFERRED

relationship where one entity inherits all attributes and constraints from another entity, expressed in EXPRESS with the SUBTYPE OF construct

3.5 supertype relationship PREFERRED

relationship where an entity serves as a base type for one or more subtypes, providing attributes and constraints inherited by those subtypes

3.6 forward attribute navigation PREFERRED

navigation that follows an attribute from an entity to the entity it references

EXAMPLE Navigating from `product_definition` through attribute `formation` to `product_definition_formation`.

3.7 inverse attribute navigation PREFERRED

navigation that finds entities referencing the current entity through a specific attribute

EXAMPLE Finding all `product_definition_relationship` entities that reference a `product_definition` through their `related_product_definition` attribute.

3.8 constraint block PREFERRED

expression that specifies conditions that entities or attributes must satisfy during path navigation

EXAMPLE `{product_definition_relationship.name = 'definition usage'}` requires the `name` attribute to equal the specified string value.

3.9 navigation step PREFERRED

single element in a reference path representing one traversal operation (subtype, supertype, forward navigation, or inverse navigation)

3.10 path expression PREFERRED

complete reference path from one schema entity to another schema entity, composed of at least one navigation step

4. OVERVIEW

4.1 General

The EXPRESS mapping language is a formal notation for specifying reference paths between entities in EXPRESS schemas.

A reference path describes a sequence of navigation steps that traverse entity relationships in the schema, starting from one schema entity and ending at another schema entity.

There are two types of reference paths used in EXPRESS mapping:

— Entity reference paths: Define how an entity in one schema relates to an entity in another schema through a series of navigations.

— Attribute reference paths: Define how an attribute of an entity in one schema relates to an attribute of an entity in another schema through a series of navigations.

4.2 Path structure

A reference path consists of one or more navigation steps, where each step represents:

— A subtype or supertype relationship between entities;

— Forward navigation through an attribute to a referenced entity;

— Inverse navigation from a referenced entity back through an attribute;

— Optional constraints on entities or attributes.

4.3 Basic example

The following simple path shows navigation through subtype relationships:

FIGURE 1

```
land <=
stratum_feature_template_component <=
laminate_component <=
assembly_component <=
component_definition <=
product_definition
```

This path indicates that `land` is a subtype of `stratum_feature_template_component`, which is a subtype of `laminate_component`, and so on, ultimately reaching `product_definition`.

4.4 Path with attribute navigation

More complex paths include attribute navigation:

FIGURE 2

```
product_definition <-
product_definition_relationship.related_product_definition
product_definition_relationship
product_definition_relationship.relating_product_definition ->
product_definition
```

This path:

a) Starts from `product_definition`

b) Navigates inversely through the `related_product_definition` attribute to find `product_definition_relationship` entities

c) Then navigates forward through the `relating_product_definition` attribute back to another `product_definition`

4.5 Path with constraints

Paths can include constraints to specify required conditions:

FIGURE 3

```
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
```

This constrains the path to only `product_definition_relationship` entities where the `name` attribute equals `'definition usage'`.

4.6 Role in ISO 10303 modules

In ISO 10303 application modules, mapping files specify how each ARM entity and attribute maps to MIM entities. These mappings appear in YAML files with the following structure:

FIGURE 4

```
ae:  # Application elements
```

```
    - entity: ARM_Entity_Name
      aimelt:
        content: mim_entity_name
      refpath:
        content: |-
          {reference path here}
      aa:  # Attribute assertions
        - attribute: attribute_name
          assertion_to: Target_ARM_Entity
          refpath:
            content: |-
              {attribute reference path here}
```

The `refpath.content` fields contain reference paths expressed in the mapping language specified by this document.

5. OPERATORS

5.1 General

5.1.1 Description

The mapping language uses operators to express different types of navigation through EXPRESS schema entity relationships.

Each operator has specific semantics defining how entities and attributes relate in the path traversal.

The operators are described in the sections below.

TABLE 1 — SUMMARY OF OPERATORS

Operator	Description	Example
`=`	Subtype relationship (left is subtype of right)	`entity_a = entity_b`
`=>`	Supertype relationship (left is supertype of right)	`entity_a => entity_b`
`-`	Inverse relationship (navigate backwards)	`entity_a - entity_b.attr`
`->`	Forward relationship (navigate through attribute)	`entity_a.attr -> entity_b`
`{…}`	Constraint block	`{entity.attr = 'value'}`
`[…]`	Required section constraint	`[section1][section2]`
`(…)`	Alternative section	`(alt1)(alt2)`
`…`	Required reference path	`path`
`|…|`	Supertype entity marker	`||supertype_entity||`
`[i]`	Aggregate element access	`attr[i]`
`[n]`	Ordered aggregate nth element	`attr[n]`
`=`	Equality constraint	`attr = 'value'`

TABLE 1 *(CONTINUED)*

Operator	Description	Example
`\`	Line continuation	`entity \`
`*`	Relationship tree structure	`*relationship*`
`--`	Comment	`-- this is a comment`
`*`	Select/enumeration extension	`select_a * select_b`
`*`	Inverse select/enumeration extension	`select_a * select_b`
`!{…}`	Negative constraint	`!{entity.attr = 'bad'}`

5.2 Subtype operator (symbol: =)

5.2.1 Description

The subtype operator = indicates that the left entity is a subtype of the right entity.

5.2.2 Syntax pattern

FIGURE 5

```
entity_a = entity_b
```

5.2.3 Semantics

— `entity_a` must be defined as a subtype of `entity_b` in the EXPRESS schema

— `entity_a` inherits all attributes and constraints from `entity_b`

— The path continues from `entity_b`

5.2.4 Example

EXAMPLE

```
land =
stratum_feature_template_component
```

Indicates that `land` is a subtype of `stratum_feature_template_component`.

5.3 Supertype operator (symbol: =>)

5.3.1 Description

The supertype operator => indicates that the left entity is a supertype of the right entity.

5.3.2 Syntax pattern

FIGURE 6

```
entity_a pass:[=] entity_b
```

5.3.3 Semantics

- `entity_a` must be defined as a supertype of `entity_b` in the EXPRESS schema
- `entity_b` is a subtype of `entity_a` and inherits from it
- The path continues from `entity_b`

5.3.4 Example

EXAMPLE

```
product_definition pass:[=]
part_template_definition
```

Indicates that `product_definition` is a supertype of `part_template_definition`.

5.4 Forward navigation operator (symbol: ->)

5.4.1 Description

The forward navigation operator `->` navigates from an entity through one of its attributes to the entity referenced by that attribute.

5.4.2 Syntax pattern

FIGURE 7

```
entity_a.attribute_name pass:[-] entity_b
```

5.4.3 Semantics

- `entity_a` must have an attribute named `attribute_name`
- The attribute must reference `entity_b` or a supertype of `entity_b`
- The path continues from `entity_b`

5.4.4 Example

EXAMPLE

```
product_definition_relationship.relating_product_definition -
product_definition
```

Navigates from `product_definition_relationship` through its `relating_product_definition` attribute to `product_definition`.

5.5 Inverse navigation operator (symbol: –)

5.5.1 Description

The inverse navigation operator – navigates backwards from a referenced entity to entities that reference it through a specific attribute.

5.5.2 Syntax pattern

FIGURE 8

```
entity_b -
entity_a.attribute_name
entity_a
```

5.5.3 Semantics

— `entity_a` must have an attribute named `attribute_name`

— The attribute must reference `entity_b` or a supertype of `entity_b`

— The path finds all `entity_a` instances that reference the current `entity_b` through `attribute_name`

— The path continues from `entity_a`

5.5.4 Example

EXAMPLE

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
```

Finds all `product_definition_relationship` entities that reference `product_definition` through their `related_product_definition` attribute.

5.6 Constraint block operators

5.6.1 Description

5.6.2 General

Constraint blocks use curly braces `{}` to specify conditions that must be satisfied during path navigation.

5.6.3 Constraint start (operator: `{`)

The constraint start operator `{` begins a constraint block.

5.6.4 Syntax pattern

FIGURE 9

```
{constraint_expression}
```

5.6.5 Constraint end (operator: `}`)

The constraint end operator `}` ends a constraint block.

5.6.6 Equality operator (symbol: `=`)

The equality operator `=` specifies that an attribute must equal a specific value.

5.6.7 Syntax pattern

FIGURE 10

```
{entity_name
entity_name.attribute_name = value}
```

Where `value` is:

— A string literal in single quotes for STRING attributes

— A numeric literal without quotes for INTEGER, REAL, or NUMBER attributes

— A boolean or logical literal (TRUE, FALSE, UNKNOWN) for BOOLEAN or LOGICAL attributes

— An enumeration value for ENUMERATION attributes

5.6.8 Example

EXAMPLE

```
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
```

Constrains the path to `product_definition_relationship` entities where the `name` attribute equals `'definition usage'`.

5.7 Alternative section operators

5.7.1 Description

5.7.2 Required sections (operator: `[…]`)

Square brackets `[]` enclose sections that must all be satisfied.

5.7.3 Syntax pattern

FIGURE 11

```
[section_1]
[section_2]
```

5.7.4 Semantics

— All enclosed sections must be satisfied

— Each section is a distinct path segment

5.7.5 Alternative sections (operator: `(…)`)

Parentheses `()` enclose alternative sections where one alternative must be satisfied.

5.7.6 Syntax pattern

FIGURE 12

```
(alternative_1)
(alternative_2)
```

5.7.7 Semantics

— Exactly one of the enclosed alternatives must be satisfied

— Each alternative represents a different possible path

5.8 Aggregate access operators

5.8.1 Description

5.8.2 Element access operator (symbol: `[i]`)

Square brackets with `i` `[i]` indicate access to any element of an aggregate attribute.

5.8.3 Syntax pattern

FIGURE 13

```
entity_name.aggregate_attribute[i]
```

5.8.4 Semantics

— The attribute must be an aggregate type (ARRAY, LIST, SET, or BAG)

— The notation accesses an arbitrary element

— The specific element accessed is not specified

5.8.5 Ordered element access operator (symbol: `[n]`)

Square brackets with a number `[n]` indicate access to a specific element of an ordered aggregate.

5.8.6 Syntax pattern

FIGURE 14

```
entity_name.aggregate_attribute[n]
```

Where `n` is a positive integer.

5.8.7 Semantics

— The attribute must be an ordered aggregate type (ARRAY or LIST)

— The notation accesses the `n`-th element

— The first element is at position 1

5.9 Special operators

5.9.1 Description

5.9.2 Negation operator (symbol: `!`)

The negation operator `!` negates a constraint.

5.9.3 Syntax pattern

FIGURE 15

```
!{entity_name
entity_name.attribute_name = value}
```

5.9.4 Semantics

- The constraint is inverted
- The path excludes entities that match the specified condition

5.9.5 Line continuation operator (symbol: `\`)

The backslash `\` indicates that the path expression continues on the next line.

5.9.6 Syntax pattern

FIGURE 16

```
long_entity_name_a \
long_entity_name_b
```

5.9.7 Semantics

- Purely syntactic
- The path is treated as if the line break did not exist
- Used for readability in complex paths

5.9.8 Comment operator (symbol: `--`)

The double dash `--` introduces a comment.

5.9.9 Syntax pattern

FIGURE 17

```
entity_name  -- this is a comment
next_entity_name
```

5.9.10 Semantics

— All text from `--` to the end of the line is ignored

— Used for documentation and annotations

— Does not affect path semantics

5.9.11 Select extension operators (symbol: `*`)

The select extension operator `*` indicates that a SELECT type extends another SELECT type.

5.9.12 Syntax pattern

FIGURE 18

```
select_type_a * select_type_b
```

5.9.13 Semantics

— `select_type_a` is extended to include all options from `select_type_b`

— Used in schema extension contexts

The inverse select extension operator `*` indicates the inverse relationship.

5.9.14 Syntax pattern

FIGURE 19

```
select_type_a * select_type_b
```

5.9.15 Semantics

— `select_type_a` is an extension of `select_type_b`

— Equivalent to `select_type_b * select_type_a`

5.9.16 Supertype entity marker (symbol: `||`)

The supertype entity marker `||` encloses a supertype entity name.

5.9.17 Syntax pattern

FIGURE 20

```
||supertype_entity||
```

5.9.18 Semantics

— Explicitly marks an entity as a supertype in the path

— Used for clarity in complex inheritance hierarchies

5.9.19 Relationship tree marker (symbol: `*`)

The relationship tree marker `*` indicates a relationship tree structure.

5.9.20 Syntax pattern

FIGURE 21

```
*relationship_entity*
```

5.9.21 Semantics

— Multiple instances of the relationship entity may be assembled in a tree

— The path between the relationship entity and related entities is implied

— Used for complex recursive relationships

5.9.22 Required reference path marker (symbol:)

The required reference path marker encloses a required path segment.

5.9.23 Syntax pattern

FIGURE 22

```
path_segment
```

5.9.24 Semantics

- The enclosed path segment must be present
- Used to emphasize mandatory portions of alternative paths

6. NAVIGATION PATTERNS

6.1 General

This clause describes common navigation patterns used in EXPRESS mapping reference paths.

These patterns represent typical ways ARM entities and attributes map to MIM entities through schema relationships.

6.2 Subtype chain navigation

6.2.1 Pattern description

A subtype chain navigates through multiple subtype relationships from a starting entity to a base entity.

6.2.2 Syntax pattern

FIGURE 23

```
subtype_entity_1 =
subtype_entity_2 =
...
base_entity
```

6.2.3 Semantics

- Each entity is a subtype of the next entity in the chain
- The chain terminates at a base entity that is not itself a subtype
- Each step must be a valid subtype relationship in the EXPRESS schema

6.2.4 Example

EXAMPLE

```
contact_size_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition
```

This path shows that `contact_size_dependent_land` ultimately derives from `product_definition` through a chain of subtypes.

6.3 Supertype chain navigation

6.3.1 Pattern description

A supertype chain navigates through multiple supertype relationships from a base entity to increasingly specific subtypes.

6.3.2 Syntax pattern

FIGURE 24

```
base_entity =
supertype_entity_1 =
supertype_entity_2 =
...
specific_subtype
```

6.3.3 Semantics

— Each entity is a supertype of the next entity in the chain

— The chain starts from a base entity and moves to more specific subtypes

— Each step must be a valid supertype relationship in the EXPRESS schema

6.3.4 Example

EXAMPLE

```
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template
```

This path shows progressive specialization from `product_definition` to the specific `land_physical_template` entity.

6.4 Forward attribute navigation

6.4.1 Pattern description

Forward attribute navigation follows an attribute from an entity to the entity it references.

6.4.2 Syntax pattern

FIGURE 25

```
source_entity.reference_attribute - target_entity
```

6.4.3 Semantics

— `source_entity` has an attribute named `reference_attribute`

— The attribute type is `target_entity` or a SELECT type containing `target_entity`

— The path continues from `target_entity`

6.4.4 Example

EXAMPLE

```
product_definition_relationship.relating_product_definition -
product_definition
```

Navigates from a `product_definition_relationship` to the `product_definition` it relates through the `relating_product_definition` attribute.

6.5 Inverse attribute navigation

6.5.1 Pattern description

Inverse attribute navigation finds entities that reference the current entity through a specific attribute.

6.5.2 Syntax pattern

FIGURE 26

```
referenced_entity -
referencing_entity.reference_attribute
```

```
    referencing_entity
```

6.5.3 Semantics

- `referencing_entity` has an attribute named `reference_attribute`
- The attribute references `referenced_entity`
- The path finds all `referencing_entity` instances referencing the current `referenced_entity`
- The path continues from `referencing_entity`

6.5.4 Example

EXAMPLE

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
```

Finds all `product_definition_relationship` entities that reference a specific `product_definition` through their `related_product_definition` attribute.

6.6 Round-trip navigation

6.6.1 Pattern description

Round-trip navigation combines inverse and forward navigation to traverse a relationship entity connecting two entities.

6.6.2 Syntax pattern

FIGURE 27

```
entity_a -
relationship_entity.attribute_to_a
relationship_entity
relationship_entity.attribute_to_b -
entity_b
```

6.6.3 Semantics

- Starts from `entity_a`
- Finds all `relationship_entity` instances referencing `entity_a`
- From those instances, navigates to `entity_b`
- Effectively finds all `entity_b` related to `entity_a` through `relationship_entity`

6.6.4 Example

EXAMPLE

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
product_definition_relationship.relating_product_definition -
product_definition
```

Finds all `product_definition` entities related to the starting `product_definition` through `product_definition_relationship`.

6.7 Constrained navigation

6.7.1 Pattern description

Constrained navigation adds conditions that entities or attributes must satisfy during path traversal.

6.7.2 Syntax pattern

FIGURE 28

```
entity_name
{entity_name
entity_name.attribute_name = constraint_value}
```

6.7.3 Semantics

— The constraint applies to the current entity in the path

— Only entities satisfying the constraint are considered valid

— Multiple constraints may be specified for the same entity

6.7.4 Example

EXAMPLE

```
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
```

Constrains the path to only those `product_definition_relationship` entities where the `name` attribute equals `'definition usage'`.

6.8 Constrained round-trip navigation

6.8.1 Pattern description

Combines round-trip navigation with constraints on the relationship entity.

6.8.2 Syntax pattern

FIGURE 29

```
entity_a -
relationship_entity.attribute_to_a
relationship_entity
{relationship_entity
relationship_entity.constraint_attribute = constraint_value}
relationship_entity.attribute_to_b -
entity_b
```

6.8.3 Semantics

— Navigates from `entity_a` through `relationship_entity` to `entity_b`

— Only `relationship_entity` instances satisfying the constraint are traversed

— Common pattern for relationship entities with type or role attributes

6.8.4 Example

EXAMPLE

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition
```

Finds related `product_definition` entities specifically through `product_definition_relationship` instances of type `'definition usage'`.

6.9 Mixed subtype and attribute navigation

6.9.1 Pattern description

Combines subtype relationships with attribute navigation in a single path.

6.9.2 Syntax pattern

FIGURE 30

```
entity_a =
entity_b =
...
base_entity -
relationship.attribute
relationship
relationship.other_attribute -
target_entity
```

6.9.3 Semantics

— First establishes the entity hierarchy through subtype relationships

— Then navigates through attributes and relationships

— Common in complex mappings involving both inheritance and association

6.9.4 Example

EXAMPLE

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition
```

Establishes the subtype chain, navigates through relationships, and continues to supertypes.

6.10 Alternative path navigation

6.10.1 Pattern description

Specifies multiple alternative paths, where at least one must be valid.

6.10.2 Syntax pattern

FIGURE 31

```
(path_alternative_1)
(path_alternative_2)
```

6.10.3 Semantics

— Exactly one of the alternatives must be satisfied

— Each alternative is a complete path segment

— Used when different schema patterns can satisfy the same mapping

6.10.4 Example

EXAMPLE

```
 (product_definition -
product_definition_formation.of_product
product_definition_formation)
 (product_definition -
product_definition_context_association.definition
product_definition_context_association)
```

The target can be reached through either `product_definition_formation` or `product_definition_context_association`.

6.11 Required combined path navigation

6.11.1 Pattern description

Specifies multiple path segments that must all be satisfied.

6.11.2 Syntax pattern

FIGURE 32

```
[required_segment_1]
[required_segment_2]
```

6.11.3 Semantics

— All required segments must be satisfied

— Each segment is a distinct portion of the path

— Used when multiple relationships must exist simultaneously

6.11.4 Example

EXAMPLE
```
[product_definition =
characterized_product_definition =
characterized_definition]
[product_definition_shape =
property_definition]
```

Both the product definition characterization and the shape property definition must exist.

7. CONSTRAINTS

7.1 General

Constraints specify conditions that entities or attributes must satisfy during path navigation.

They enable precise mapping specifications by limiting paths to entities matching specific criteria.

7.2 Constraint syntax

7.2.1 Basic structure

A constraint block is enclosed in curly braces and contains one or more constraint expressions.

Syntax:

FIGURE 33

```
{entity_name
entity_name.attribute_name operator value}
```

7.2.2 Multiple constraints

Multiple constraints may be specified within a single block or across multiple blocks.

Syntax for single block:

FIGURE 34

```
{entity_name
entity_name.attribute_1 = value_1
entity_name.attribute_2 = value_2}
```

Syntax for multiple blocks:

FIGURE 35

```
{entity_name
entity_name.attribute_1 = value_1}
{entity_name
entity_name.attribute_2 = value_2}
```

Semantics:

— All constraints must be satisfied

— Multiple constraints represent a logical AND condition

EXAMPLE

```
{product_definition_relationship
product_definition_relationship.name = 'definition usage'
product_definition_relationship.description = 'explicit'}
```

Requires both the `name` and `description` attributes to match.

7.3 Value types

7.3.1 String values

String values must be enclosed in single quotes.

Syntax:

FIGURE 36

```
{entity_name
entity_name.string_attribute = 'string value'}
```

EXAMPLE

```
{product_definition_relationship
```

```
    product_definition_relationship.name = 'definition usage'}
```

7.3.2 Numeric values

Numeric values (INTEGER, REAL, NUMBER) are specified without quotes.

Syntax:

FIGURE 37

```
{entity_name
entity_name.numeric_attribute = numeric_value}
```

EXAMPLE

```
    {dimension_count
    dimension_count.dim = 3}
```

7.3.3 Boolean and logical values

Boolean and logical values use keywords TRUE, FALSE, or UNKNOWN.

Syntax:

FIGURE 38

```
{entity_name
entity_name.boolean_attribute = TRUE}
```

EXAMPLE

```
    {representation_item
    representation_item.optional_flag = TRUE}
```

7.3.4 Enumeration values

Enumeration values are specified as identifiers without quotes.

Syntax:

FIGURE 39

```
{entity_name
entity_name.enum_attribute = ENUM_VALUE}
```

EXAMPLE

```
    {geometric_representation_context
```

```
geometric_representation_context.coordinate_space_dimension = THREE_D}
```

7.4 Negated constraints

7.4.1 Syntax

The negation operator `!` inverts a constraint.

FIGURE 40

```
!{entity_name
entity_name.attribute_name = value}
```

7.4.2 Semantics

— The path excludes entities matching the specified condition

— Equivalent to “attribute_name NOT EQUAL TO value”

7.4.3 Example

EXAMPLE

```
!{product_definition_relationship
product_definition_relationship.name = 'alternate'}
```

Excludes `product_definition_relationship` entities where `name` is `'alternate'`.

7.5 Constraint application

7.5.1 Applying to current entity

Constraints apply to the entity specified in the constraint block.

FIGURE 41

```
entity_name
{entity_name
entity_name.attribute = value}
```

The constraint is checked when the path reaches `entity_name`.

7.5.2 Applying during navigation

Constraints can be placed at any point in the path to filter entities during navigation.

EXAMPLE

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition
```

The constraint filters which `product_definition_relationship` instances are traversed.

7.6 Constraint validation

7.6.1 Entity validation

The constrained entity must:

— Exist in the EXPRESS schema

— Appear in the path at the location where the constraint is specified

7.6.2 Attribute validation

The constrained attribute must:

— Exist on the specified entity

— Be accessible (not private or restricted)

— Have a type compatible with the constraint value

7.6.3 Value validation

The constraint value must:

— Match the attribute's base type (STRING, INTEGER, etc.)

— Follow proper literal syntax for the type

— For enumerations, be a valid enumeration value

7.6.4 Type compatibility

The following table shows required value syntax for each attribute type:

TABLE 2

Attribute type	Value syntax	Example
STRING	Single-quoted string	`'definition usage'`
INTEGER	Unquoted integer	`42`
REAL	Unquoted decimal	`3.14159`
NUMBER	Unquoted number	`100` or `2.5`
BOOLEAN	TRUE or FALSE	`TRUE`

(CONTINUED)

Attribute type	Value syntax	Example
LOGICAL	TRUE, FALSE, or UNKNOWN	`UNKNOWN`
ENUMERATION	Unquoted enumeration value	`CARTESIAN`

7.7 Common constraint patterns

7.7.1 Relationship type constraints

Constraining relationship entities by their type or role.

EXAMPLE

```
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
```

7.7.2 Geometry type constraints

Constraining geometric entities by their dimensionality or type.

EXAMPLE

```
{geometric_representation_context
geometric_representation_context.coordinate_space_dimension = 3}
```

7.7.3 Description constraints

Constraining by description attributes that specify roles or purposes.

EXAMPLE

```
{shape_aspect
shape_aspect.description = 'land interface terminal'}
```

7.7.4 Multiple attribute constraints

Requiring multiple attributes to satisfy conditions simultaneously.

EXAMPLE

```
{characterized_object
characterized_object.name = 'boundary'
characterized_object.description = 'outer'}
```

8. VALIDATION RULES

8.1 General

Validation ensures that reference paths are correct and navigate valid relationships in the EXPRESS schema repository.

All validation rules shall be applied to verify path correctness.

8.2 Entity validation

8.2.1 Entity existence

All entities referenced in a path must exist in the EXPRESS schema repository.

Rule:

— For each entity name in the path, the validator shall verify that an entity with that exact name exists in the repository

Error condition:

— If an entity does not exist, the path is invalid

Error message shall include:

— The entity name that was not found

— The location in the path where the entity appears

— The source file and line number if available

EXAMPLE For the path:

```
non_existent_entity =
product_definition
```

If `non_existent_entity` does not exist in the schema, the validator shall report: “Entity ‘non_existent_entity’ not found in repository”.

8.2.2 Entity name case sensitivity

Entity names in EXPRESS are case-insensitive, but the validator should use canonical case matching.

Rule:

— Entity name matching shall be case-insensitive

— The validator may optionally warn if case does not match the schema definition

8.3 Relationship validation

8.3.1 Subtype relationship validation

For subtype operator =, the left entity must be a declared subtype of the right entity.

Rule:

— The validator shall verify that the left entity's definition includes the right entity in its SUBTYPE OF clause, either directly or through transitive subtype relationships

Error condition:

— If the subtype relationship does not exist, the path is invalid

Error message shall include:

— The two entity names involved

— Whether the relationship should be direct or transitive

— A suggestion to verify the entity hierarchy

EXAMPLE For the path:

```
land =
product_definition
```

The validator shall verify that `land` is a subtype of `product_definition`, either directly or through intermediate entities.

8.3.2 Supertype relationship validation

For supertype operator =, the left entity must be a declared supertype of the right entity.

Rule:

— The validator shall verify that the right entity's definition includes the left entity in its SUBTYPE OF clause, either directly or through transitive relationships

Error condition:

— If the supertype relationship does not exist, the path is invalid

8.4 Attribute validation

8.4.1 Attribute existence

All attributes referenced in navigation must exist on the specified entity.

Rule:

— For forward navigation `entity.attribute` →, the validator shall verify that `entity` has an attribute named `attribute`

— For inverse navigation `← entity.attribute`, the validator shall verify that `entity` has an attribute named `attribute`

Error condition:

— If the attribute does not exist, the path is invalid

Error message shall include:

— The entity name

— The attribute name that was not found

— Available attributes on the entity (as suggestion)

8.4.2 Attribute name case sensitivity

Attribute names in EXPRESS are case-insensitive.

Rule:

— Attribute name matching shall be case-insensitive

— The validator may optionally warn if case does not match the schema definition

8.5 Navigation validation

8.5.1 Forward navigation validation

For forward navigation `entity_a.attribute → entity_b`:

Rules:

— The attribute must exist on `entity_a`

— The attribute type must be `entity_b` or a type that can reference `entity_b`

— If the attribute type is a SELECT, `entity_b` must be one of the SELECT options

Error conditions:

— Attribute does not exist on `entity_a`

— Attribute type is not compatible with `entity_b`

— Attribute type is a basic type (STRING, INTEGER, etc.) and cannot reference an entity

8.5.2 Inverse navigation validation

For inverse navigation `entity_b ← entity_a.attribute`:

Rules:

— The attribute must exist on `entity_a`

— The attribute type must be `entity_b` or a type that can reference `entity_b`

— The path is following the relationship backwards from `entity_b` to `entity_a`

Error conditions:

— Attribute does not exist on `entity_a`

— Attribute does not reference `entity_b`

8.5.3 Navigation continuity

The validator shall verify that each navigation step properly connects to the next step.

Rule:

— The ending entity of one step must match or be compatible with the starting entity of the next step

— For subtype/supertype chains, entities must be properly related

— For attribute navigation, the referenced entity must match the next entity in the path

Error condition:

— If navigation continuity is broken, the path is invalid

8.6 Constraint validation

8.6.1 Constraint entity validation

For each constraint, the constrained entity must:

— Exist in the schema

— Appear in the path at or before the constraint location

Rule:

— The validator shall verify that the entity specified in the constraint block exists and is part of the current path context

8.6.2 Constraint attribute validation

For each constraint, the constrained attribute must:

— Exist on the specified entity

— Be accessible (not DERIVED or INVERSE unless explicitly allowed)

Rule:

— The validator shall verify that the attribute exists on the entity and can be constrained

Error condition:

— If the attribute does not exist or cannot be constrained, the constraint is invalid

8.6.3 Constraint value type validation

For each constraint value, the type must be compatible with the attribute type.

Rules for type compatibility:

TABLE 3

Attribute type	Required value syntax	Validation rule
STRING	Single-quoted string	Value must be a valid string literal
INTEGER	Unquoted integer	Value must be a valid integer
REAL	Unquoted decimal	Value must be a valid real number
NUMBER	Unquoted number	Value must be a valid number
BOOLEAN	TRUE or FALSE	Value must be TRUE or FALSE
LOGICAL	TRUE, FALSE, or UNKNOWN	Value must be TRUE, FALSE, or UNKNOWN
ENUMERATION	Unquoted identifier	Value must be a valid enumeration item
SELECT	Depends on selected type	Value must match one of the SELECT options

Error condition:

— If value type does not match attribute type, the constraint is invalid

Error message shall include:

— The attribute name and its type

— The provided value and its inferred type

— Expected value format

EXAMPLE For a STRING attribute:

```
{entity.string_attr = 'valid'}     -- Valid
{entity.string_attr = invalid}     -- Invalid: missing quotes
{entity.string_attr = 123}         -- Invalid: wrong type
```

8.7 Path completeness validation

8.7.1 Start entity validation

The path should start with a valid entity from the ARM entity being mapped.

Rule:

— The first entity in the refpath should match or be compatible with the ARM entity specified in the mapping

Note: This rule may be relaxed for attribute mappings where the path starts from a different context.

8.7.2 End entity validation

The path should end with a valid MIM entity.

Rule:

— The final entity in the path should be a valid MIM entity that can represent the ARM concept

Note: The specific MIM entity expected may be specified in the mapping's `aimelt` field.

8.8 Error collection

The validator shall collect all errors rather than stopping at the first error.

Rule:

— The validator shall continue processing the entire path

— All errors shall be collected and reported together

— Each error shall include sufficient context for diagnosis

8.9 Warning conditions

The validator may issue warnings for conditions that are not strictly invalid but may indicate problems:

— Case mismatches between path and schema definitions

— Deprecated entities or attributes

— Excessively long paths that may indicate design issues

— Redundant navigation steps

— Constraints that are always true or always false

8.10 Validation output

8.10.1 Success output

For valid paths, the validator shall report:

— Path validation status: VALID

— Starting entity

— Ending entity

— Number of navigation steps

- List of entities traversed

8.10.2 Error output

For invalid paths, the validator shall report:

- Path validation status: INVALID
- List of all errors found
- For each error:
  - Error type (entity not found, invalid relationship, etc.)
  - Error message
  - Location in path (entity name, line number if available)
  - Relevant path context
  - Suggestion for correction if available

8.10.3 Output formats

The validator should support multiple output formats:

- Text format for human reading
- JSON format for programmatic processing
- YAML format for integration with other tools

ANNEX A
(NORMATIVE)

EXAMPLES

A.1 General

This annex provides comprehensive examples of EXPRESS mapping language reference paths drawn from actual ISO 10303 module mapping specifications. Each example demonstrates specific patterns and operators described in the normative clauses of this document.

A.2 Simple entity chains

A.2.1 General

Simple entity chains demonstrate the most basic form of reference path navigation, using only subtype and supertype operators.

A.2.2 Basic subtype chain

This example from ISO 10303-1692 (Land) shows a simple subtype chain navigating from a specialized entity to its base entity.

EXAMPLE — Simple subtype navigation

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition
```

This path specifies that:

— `land` is a subtype of `stratum_feature_template_component`

— `stratum_feature_template_component` is a subtype of `laminate_component`

— `laminate_component` is a subtype of `assembly_component`

— `assembly_component` is a subtype of `component_definition`

— `component_definition` is a subtype of `product_definition`

The path establishes the complete subtype hierarchy from the ARM entity `Land` to the base MIM entity `product_definition`.

A.2.3 Extended subtype chain

This example shows a longer subtype chain for more specialized entities.

EXAMPLE — Extended subtype navigation

```
plated_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition
```

This path demonstrates:

— Multiple levels of specialization through subtype relationships

— Navigation from a highly specialized entity (`plated_passage_dependent_land`) through intermediate specializations to the base entity

— The hierarchical nature of entity classifications in ISO 10303

A.3 Round-trip navigation

A.3.1 General

Round-trip paths navigate forward through relationships, then backward through inverse relationships, demonstrating complex attribute-based navigation patterns.

A.3.2 Simple round-trip with constraint

This example from ISO 10303-1692 shows a complete round-trip pattern with a constraint.

EXAMPLE — Round-trip with constrained relationship

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template
```

This path demonstrates:

- Subtype chain navigation (⇐) from `land` to `product_definition`
- Inverse attribute navigation (←) via `product_definition_relationship.related_product_definition`
- Constraint application ({}) requiring `name = 'definition usage'`
- Forward attribute navigation (→) via `product_definition_relationship.relating_product_definition`
- Supertype chain navigation (⇒) from `product_definition` to `land_physical_template`

The complete path maps the ARM attribute `derived_from` of entity `Land` to an instance of `Land_physical_template` in the MIM.

A.3.3 Complex round-trip with multiple constraints

This example shows a more complex round-trip with detailed constraints.

EXAMPLE — Complex constrained round-trip

```
plated_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template =
default_passage_based_land_physical_template =
default_plated_passage_based_land_physical_template
```

This path illustrates:

- A complete subtype chain to establish the starting context
- A constrained relationship traversal to find related definitions
- An extended supertype chain navigating through multiple specialization levels
- Multiple constraint validations ensuring the correct relationship type

The path maps from a specialized ARM entity through a constrained relationship to a specialized template in the MIM.

A.4 Shape aspect navigation

A.4.1 General

Shape aspect navigation patterns demonstrate complex traversals through shape definitions and feature relationships.

A.4.2 Shape aspect with feature instantiation

This example from ISO 10303-1698 shows navigation through shape aspects and feature relationships.

EXAMPLE — Shape aspect to component navigation

```
laminate_component_interface_terminal =
laminate_component_feature =
component_feature =
shape_aspect
shape_aspect.of_shape -
product_definition_shape =
property_definition
property_definition.definition -
characterized_definition
characterized_definition = characterized_product_definition
characterized_product_definition
characterized_product_definition = product_definition
product_definition =
component_definition =
assembly_component =
laminate_component =
stratum_feature_template_component =
land =
contact_size_dependent_land
```

This path demonstrates:

— Subtype navigation through shape aspect hierarchy

— Forward navigation via `shape_aspect.of_shape` attribute

— Navigation through property definitions

— Type selection using `=` operator for `characterized_definition`

— Multiple type selections for `characterized_product_definition`

— Supertype chain to reach the target entity

The complete path establishes the relationship between a terminal feature and its associated component.

A.4.3 Shape aspect with relationship constraint

This example shows constrained navigation through shape aspect relationships.

EXAMPLE — Constrained shape aspect relationship

```
laminate_component_interface_terminal =
laminate_component_feature =
component_feature =
shape_aspect -
shape_aspect_relationship.related_shape_aspect
shape_aspect_relationship
{shape_aspect_relationship
shape_aspect_relationship.name = 'instantiated feature'}
shape_aspect_relationship.relating_shape_aspect -
{shape_aspect
shape_aspect.description = 'interface terminal'}
shape_aspect =
land_template_terminal
```

This path illustrates:

- Inverse navigation through `shape_aspect_relationship`
- Constraint on the relationship name
- Forward navigation to the relating shape aspect
- Constraint on the shape aspect description
- Supertype navigation to the target entity

The path validates both the relationship type and the shape aspect characteristics.

A.5 Alternative paths

A.5.1 General

Alternative paths specify multiple valid navigation routes to reach the target entity or attribute value.

A.5.2 Simple alternative with shared endpoint

This example from ISO 10303-1692 demonstrates alternative subtype chains.

EXAMPLE — Alternative subtype paths

```
land_with_join_terminal =
[land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition]
[laminate_component_join_terminal =
laminate_component_feature =
component_feature =
shape_aspect]
```

This path specifies:

- Two alternative navigation routes enclosed in `[]` brackets
- First alternative: subtype chain through product definition hierarchy
- Second alternative: subtype chain through shape aspect hierarchy

— Both alternatives must lead to valid entity instances

The entity `land_with_join_terminal` can be reached through either path.

A.6 Constrained paths

A.6.1 General

Constrained paths apply validation criteria at specific points in the navigation sequence.

A.6.2 Single constraint on entity attribute

This example shows a simple constraint on an entity attribute.

EXAMPLE — Attribute value constraint

```
laminate_component_interface_terminal =
laminate_component_feature =
component_feature =
shape_aspect
{shape_aspect
shape_aspect.description = 'land interface terminal'}
```

This path demonstrates:

— Navigation through a subtype chain

— Constraint application at the final entity

— Validation that `description` attribute equals a specific string value

Only instances with the specified description value satisfy this mapping.

A.6.3 Multiple constraints in sequence

This example shows multiple constraints applied at different points.

EXAMPLE — Multiple sequential constraints

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'alternate instantiated template'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template
```

This path illustrates:

— A subtype chain leading to a relationship entity

— Constraint on the relationship's name attribute

— Navigation through the constrained relationship

— Continued navigation through supertype chain

Each constraint must be satisfied for the path to be valid.

A.7 Reference relationships

A.7.1 General

Reference relationship patterns establish connections between entities through named relationships.

A.7.2 Simple reference relationship

This example shows a basic reference relationship pattern.

EXAMPLE — Named reference relationship

```
plated_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'reference plated passage'}
product_definition_relationship.relating_product_definition -
product_definition =
component_definition =
assembly_component =
laminate_component =
inter_stratum_feature =
plated_inter_stratum_feature =
plated_passage
```

This path demonstrates:

— Establishing context through subtype chain

— Inverse navigation to find relationships

— Constraint identifying the reference relationship by name

— Forward navigation through the relationship

— Supertype navigation to the referenced entity

The pattern establishes the `reference_plated_passage` attribute mapping.

A.7.3 Multiple reference relationships

This example from ISO 10303-1692 shows handling multiple reference relationships.

EXAMPLE — Dual reference pattern

```
unsupported_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'reference passage'}
product_definition_relationship.relating_product_definition -
product_definition =
component_definition =
assembly_component =
laminate_component =
inter_stratum_feature =
unsupported_passage
```

This path illustrates:

— A reference relationship identified by name ‘reference passage’

— Navigation pattern similar to other reference relationships

— Targeting a different entity type (unsupported_passage)

Different reference relationship names distinguish between multiple relationships from the same source entity.

A.8 Structured template navigation

A.8.1 General

Structured template patterns navigate through geometric template hierarchies.

A.8.2 Structured component to template

This example from ISO 10303-1698 shows navigation from a structured component to its template.

EXAMPLE — Structured template reference

```
thermal_isolation_removal_component =
material_removal_structured_component =
structured_layout_component =
assembly_group_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship.name = 'definition usage'}
```

```
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
geometric_template =
structured_template =
single_stratum_structured_template =
material_removal_structured_template =
thermal_isolation_removal_template
```

This path demonstrates:

— Navigation through structured component hierarchy

— Constrained relationship to template definition

— Navigation through geometric template specializations

— Multiple levels of template specialization

The pattern establishes the relationship between a physical component and its abstract template definition.

A.8.3 Dependent thermal isolation pattern

This example shows a dependent component referencing its template.

EXAMPLE — Dependent component template reference

```
dependent_thermal_isolation_removal_component =
thermal_isolation_removal_component =
material_removal_structured_component =
structured_layout_component =
assembly_group_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
geometric_template =
structured_template =
single_stratum_structured_template =
material_removal_structured_template =
thermal_isolation_removal_template =
dependent_thermal_isolation_removal_template
```

This path illustrates:

— Additional level of specialization for dependent components

— Same relationship pattern as independent components

— Extended template specialization hierarchy

Dependent and independent components follow parallel mapping patterns with different specialization levels.

A.9 Material removal patterns

A.9.1 General

Material removal patterns demonstrate navigation through specialized component and template hierarchies.

A.9.2 Electrical isolation removal

This example from ISO 10303-1698 shows electrical isolation removal mapping.

EXAMPLE — Electrical isolation component to template

```
dependent_electrical_isolation_removal_component =
electrical_isolation_laminate_component =
material_removal_laminate_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
material_removal_feature_template =
electrical_isolation_removal_template =
dependent_electrical_isolation_removal_template
```

This path demonstrates:

— Material removal component specialization hierarchy

— Constrained definition usage relationship

— Material removal feature template hierarchy

— Parallel structure between component and template hierarchies

The pattern establishes the mapping for electrical isolation removal features.

A.10 Attachment size based patterns

A.10.1 General

Attachment size based patterns demonstrate navigation to templates parameterized by attachment dimensions.

A.10.2 Contact size dependent land

This example from ISO 10303-1692 shows navigation to an attachment size based template.

EXAMPLE — Attachment size based template reference

```
contact_size_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template =
default_attachment_size_based_land_physical_template
```

This path demonstrates:

- Navigation from a size-dependent component
- Constrained relationship to template definition
- Specialization to attachment size based template
- The relationship between component sizing and template selection

The pattern enables parameterization of land geometry based on attached component sizes.

A.11 Passage based patterns

A.11.1 General

Passage based patterns establish relationships between land features and passage features that they interface with.

A.11.2 Unsupported passage based land

This example shows the mapping for lands based on unsupported passages.

EXAMPLE — Unsupported passage based template

```
unsupported_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template =
default_passage_based_land_physical_template =
default_unsupported_passage_based_land_physical_template
```

This path illustrates:

- Navigation from passage-dependent land
- Template specialization for passage-based features
- Additional specialization for unsupported passages
- Hierarchical template classification

The pattern enables land geometry to be defined relative to associated passage features.

A.11.3 Plated passage based land

This example shows the mapping for lands based on plated passages.

EXAMPLE — Plated passage based template

```
plated_passage_dependent_land =
inter_stratum_feature_dependent_land =
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template =
default_passage_based_land_physical_template =
default_plated_passage_based_land_physical_template
```

This path demonstrates:

- Same base pattern as unsupported passage based land
- Different final template specialization
- Consistent mapping structure across passage types

The pattern maintains structural consistency while accommodating different passage metallization types.

A.12 Alternate definition patterns

A.12.1 General

Alternate definition patterns provide alternative template references for entities that support multiple configuration options.

A.12.2 Alternate land definition

This example from ISO 10303-1692 shows the mapping for alternate land template definitions.

EXAMPLE — Alternate template instantiation

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'alternate instantiated template'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template
```

This path demonstrates:

- Direct navigation to relationship entity (no inverse navigation)
- Constraint specifying 'alternate instantiated template' relationship
- Navigation through the relationship to template
- Simplified path for alternate definitions

The pattern enables entities to reference multiple template definitions for different configuration scenarios.

A.13 Validation scenarios

A.13.1 General

These examples illustrate various validation scenarios and error conditions.

A.13.2 Valid path with all constraints satisfied

This example shows a valid path where all constraints are satisfied.

EXAMPLE — Fully valid reference path

```
land =
stratum_feature_template_component =
laminate_component =
assembly_component =
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'definition usage'}
product_definition_relationship.relating_product_definition -
product_definition =
part_template_definition =
single_stratum_template =
single_stratum_continuous_template =
stratum_feature_template =
land_physical_template
```

Validation checks:

— All entities exist in the EXPRESS repository

— All subtype relationships are valid

— Attributes used for navigation exist

— Relationship constraints are syntactically correct

— Complete path from ARM to MIM

This path would pass all validation rules.

A.13.3 Invalid entity reference

This example shows an error condition with an invalid entity reference.

EXAMPLE — Invalid entity in path

```
land =
invalid_entity_name =
laminate_component =
assembly_component
```

Validation errors:

— Entity 'invalid_entity_name' does not exist in the EXPRESS repository

— Subtype relationship cannot be validated

— Path is incomplete

This path would fail entity existence validation.

A.13.4 Invalid attribute reference

This example shows an error condition with an invalid attribute reference.

EXAMPLE — Invalid attribute navigation

```
land =
stratum_feature_template_component =
laminate_component
laminate_component.invalid_attribute -
product_definition
```

Validation errors:

- Attribute ‘invalid_attribute’ does not exist on entity ‘laminate_component’
- Navigation cannot proceed
- Path is broken

This path would fail attribute existence validation.

A.13.5 Constraint validation failure

This example shows a constraint that cannot be validated.

EXAMPLE — Invalid constraint syntax

```
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.invalid_attribute = 'value'}
```

Validation errors:

- Attribute ‘invalid_attribute’ does not exist on ‘product_definition_relationship’
- Constraint cannot be evaluated
- Path may proceed but constraint is invalid

This path would generate a constraint validation warning.

A.13.6 Type mismatch in navigation

This example shows a type mismatch error.

EXAMPLE — Type incompatibility

```
shape_aspect
shape_aspect.of_shape -
product_definition
```

Validation errors:

- Attribute ‘of_shape’ has type ‘product_definition_shape’
- Expected target is ‘product_definition_shape’, not ‘product_definition’
- Type mismatch in navigation

This path would fail type compatibility validation.

A.14 Complex combined patterns

A.14.1 General

These examples demonstrate combinations of multiple navigation patterns in a single reference path.

A.14.2 Multi-constraint round-trip with alternatives

This hypothetical example combines multiple advanced patterns.

EXAMPLE — Complex combined navigation

```
specialized_component =
[base_component_type_a =
component_definition]
[base_component_type_b =
alternate_component_definition]
{component_definition
component_definition.description = 'specialized'}
component_definition =
product_definition -
product_definition_relationship.related_product_definition
product_definition_relationship
{product_definition_relationship
product_definition_relationship.name = 'usage'}
product_definition_relationship.relating_product_definition -
{product_definition
product_definition.description = 'template'}
product_definition =
template_definition =
specialized_template
```

This path demonstrates:

- Alternative paths for initial navigation
- Constraints on both source and target entities
- Round-trip through relationship
- Multiple constraint validations
- Combined use of all major operators

Such complex paths require careful validation at each step.

A.15 Summary of example patterns

The examples in this annex demonstrate:

a) Simple subtype and supertype chains establishing entity hierarchies

b) Round-trip navigation through forward and inverse attribute references

c) Constrained paths validating attribute values at specific points

d) Alternative paths providing multiple valid navigation routes

e) Reference relationship patterns establishing named entity connections

f) Shape aspect navigation through feature and property hierarchies

g) Template reference patterns connecting physical components to abstract definitions

h) Complex combined patterns using multiple operators and constraints

Each pattern serves specific mapping requirements in the ISO 10303 modular architecture, enabling precise specification of how ARM entities and attributes correspond to MIM entity instances and values.

ANNEX B
(INFORMATIVE)

BIBLIOGRAPHY