ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р ИСО 10303-14— 2015

Системы автоматизации производства и их интеграция

ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 14

Методы описания. Справочное руководство по языку EXPRESS-X

ISO 10303-14:2005
Industrial automation systems and integration — Product data representation and exchange — Part 14: Description methods: The EXPRESS-X language reference manual (IDT)

Издание официальное

Москва
Стандартинформ
2016

gost.e-ecolog.ru

Предисловие

1 ПОДГОТОВЛЕН Федеральным государственным автономным научным учреждением «Центральный научно-исследовательский и опытно-конструкторский институт робототехники и технической кибернетики» (ЦНИИ РТК) на основе собственного аутентичного перевода на русский язык международного стандарта, указанного в пункте 4

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 459 «Информационная поддержка жизненного цикла изделий»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 21 июля 2015 г. № 925-ст

4 Настоящий стандарт идентичен международному стандарту ИСО 10303-14:2005 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 14. Методы описания. Справочное руководство по языку EXPRESS-X» (ISO 10303-14:2005 «Industrial automation systems and integration — Product data representation and exchange — Part 14: Description methods: The EXPRESS-X language reference manual»).

При применении настоящего стандарта рекомендуется использовать вместо ссылочных международных стандартов соответствующие им национальные стандарты Российской Федерации, сведения о которых приведены в дополнительном приложении ДА

5 ВВЕДЕН ВПЕРВЫЕ

*Правила применения настоящего стандарта установлены в ГОСТ Р 1.0—2012 (раздел 8). Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (*gost.e-ecolog.ru

© Стандартинформ, 2016

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область применения . . . 1
2 Нормативные ссылки . . . 1
3 Термины и определения . . . 2
4 Основные принципы . . . 3
4.1 Введение . . . 3
4.2 Основные принципы модели исполнения . . . 4
4.3 Среда исполнения . . . 6
5 Требования соответствия . . . 6
5.1 Классы соответствия языка EXPRESS-X . . . 6
6 Синтаксис спецификации языка . . . 8
7 Основные элементы языка . . . 9
7.1 Введение . . . 9
7.2 Зарезервированные слова . . . 9
8 Типы данных . . . 9
8.1 Введение . . . 9
8.2 Тип данных образа . . . 9
9 Объявления . . . 10
9.1 Введение . . . 10
9.2 Связывание . . . 10
9.3 Объявление образа . . . 14
9.4 Объявление отображения . . . 18
9.5 Объявление образа схемы . . . 28
9.6 Объявление отображения схемы . . . 29
9.7 Локальное объявление . . . 30
9.8 Объявление констант . . . 31
9.9 Объявление функций . . . 31
9.10 Объявление процедур . . . 31
9.11 Объявление правил . . . 31
10 Выражения . . . 31
10.1 Введение . . . 31
10.2 Вызов образа . . . 32
10.3 Вызов отображения . . . 34
10.4 Вызовы частичного связывания . . . 36
10.5 Выражение FOR . . . 37
10.6 Выражение IF . . . 40
10.7 Выражение CASE . . . 40
10.8 Оператор прямого пути . . . 40
10.9 Оператор обратного пути . . . 41
11 Встроенная функция . . . 43
11.1 Универсальная функция EXTENT . . . 43
12 Области действия и видимости . . . 43
12.1 Введение . . . 43
12.2 Образ схемы . . . 44

12.3 Отображение схемы 44
12.4 Образ и зависимый образ 45
12.5 Метка раздела образа 45
12.6 Идентификатор атрибута образа 45
12.7 Выражение FOR 45
12.8 Отображение и зависимое отображение 45
12.9 Элемент языка FROM 46
12.10 Цикл создания экземпляра объекта 46
12.11 Оператор пути 46
13 Спецификация интерфейса 46
13.1 Введение 46
13.2 Элемент языка REFERENCE 46
Приложение А (обязательное) Регистрация информационного объекта 48
Приложение В (обязательное) Синтаксис языка EXPRESS-X 49
Приложение С (обязательное) Алгоритм преобразования текста с языка EXPRESS-X на язык EXPRESS 59
Приложение D (справочное) Вопросы реализации 61
Приложение Е (справочное) Функция unnest оператора пути 62
Приложение F (справочное) Семантика таблицы отображений 63
Приложение ДА (справочное) Сведения о соответствии ссылочных международных стандартов национальным стандартам Российской Федерации 67
Библиография 68

Введение

Стандарты комплекса ИСО 10303 распространяются на компьютерное представление информации об изделиях и обмен данными об изделиях. Их целью является обеспечение нейтрального механизма, способного описывать изделия на всем протяжении их жизненного цикла. Этот механизм применим не только для обмена файлами в нейтральном формате, но является также основой для реализации и совместного доступа к базам данных об изделиях и организации архивирования.

Стандарты комплекса ИСО 10303 представляют собой набор отдельно издаваемых стандартов (частей). Стандарты данного комплекса относятся к одной из следующих тематических групп: «Методы описания», «Методы реализации», «Методология и основы аттестационного тестирования», «Интегрированные обобщенные ресурсы», «Интегрированные прикладные ресурсы», «Прикладные протоколы», «Комплекты абстрактных тестов», «Прикладные интерпретированные конструкции» и «Прикладные модули». Полный перечень стандартов комплекса ИСО 10303 представлен на сайте gost.e-ecolog.ru. Настоящий стандарт входит в тематическую группу «Методы описания». Он подготовлен подкомитетом SC4 «Производственные данные» Технического комитета 184 ИСО «Системы автоматизации производства и их интеграция». Приложения A, B и C являются неотъемлемой частью настоящего стандарта; приложения D, E и F являются справочными.

Настоящий стандарт определяет язык для описания взаимосвязей между данными, управляемыми EXPRESS-схемами, и для описания альтернативных представлений таких данных. Данный язык называется EXPRESS-X.

Пользователи настоящего стандарта должны быть знакомы со спецификацией языка EXPRESS, определенной в ИСО 10303-11, и со спецификацией кодирования открытым текстом структуры обмена, определенной в ИСО 10303-21.

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Системы автоматизации производства и их интеграция

ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 14

Методы описания.
Справочное руководство по языку EXPRESS-X

Industrial automation systems and integration. Product data representation and exchange.
Part 14. Description methods. The EXPRESS-X language reference manual

Дата введения — 2016—10—01

1 Область применения

Настоящий стандарт определяет язык для описания взаимосвязей между данными, управляемыми EXPRESS-схемами, и для описания альтернативных представлений таких данных. Данный язык называется EXPRESS-X.

EXPRESS-X является языком отображения структурированных данных. Данный язык состоит из элементов, которые позволяют однозначно определять взаимосвязь между EXPRESS-схемами.

Требования настоящего стандарта распространяются:

- на отображение данных, управляемых одной EXPRESS-схемой, на данные, управляемые другой EXPRESS-схемой;
- отображение данных, управляемых одной версией EXPRESS-схемы, на данные, управляемые другой версией той же EXPRESS-схемы, причем эти две версии схемы имеют разные имена;
- определение требований к трансляторам данных для приложений совместного использования данных и обмена данными;
- спецификацию альтернативных представлений данных, определенных в EXPRESS-схеме;
- альтернативное представление таблиц отображения прикладных протоколов;
- двунаправленные отображения в тех случаях, когда это математически возможно;
- спецификацию ограничений, которые могут быть оценены по отношению к данным, полученным в результате отображения.

Требования настоящего стандарта не распространяются:

- на отображение данных, определенных с помощью средств, отличных от языка EXPRESS;
- идентификацию версии EXPRESS-схемы;
- графическое представление конструкций языка EXPRESS-X.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие международные стандарты (для датированных ссылок следует использовать только указанное издание, для недатированных ссылок — последнее издание указанного документа, включая все поправки к нему):

ИСО/МЭК 8824-1:2002* Информационная технология. Абстрактная синтаксическая нотация версии 1 (АСН.1). Часть 1. Спецификация основной нотации (ISO/IEC 8824-1:2002, Information technology — Abstract Syntax Notation One (ASN.1) — Part 1: Specification of basic notation)

* Отменен. Действует ИСО/МЭК 8824-1:2008. Для однозначного соблюдения требований настоящего стандарта, выраженных в датированных ссылках, рекомендуется использовать только данный ссылочный стандарт.

Издание официальное

ИСО 10303-1:1994 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы (ISO 10303-1:1994, Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles)

ИСО 10303-11:2004 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 11. Методы описания. Справочное руководство по языку EXPRESS (ISO 10303-11:2004, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual)

ИСО/МЭК 10646-1:2000* Информационные технологии. Универсальный многооктетный набор кодированных символов (UCS). Часть 1. Архитектура и основная многоязычная матрица (ISO/IEC 10646:2003, Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multiligual Plane)

3 Термины и определения

3.1 Термины, определенные в ИСО 10303-1

В настоящем стандарте применены следующие термины:

- **данные** (data);
- **информация** (information).

3.2 Термины, определенные в ИСО/МЭК 10303-11

В настоящем стандарте применены следующие термины:

- **сложный объектный тип данных** (complex entity data type);
- **экземпляр сложного объекта (сложного объектного типа данных)** [complex entity (data type) instance];
- **константа** (constant);
- **объект** (entity);
- **объектный тип данных** (entity data type);
- **экземпляр объекта (объектного типа данных)** [entity (data type) instance];
- **экземпляр** (instance);
- **частичный сложный объектный тип данных** (partial complex entity data type);
- **значение частичного сложного объекта** (partial complex entity value);
- **совокупность** (population);
- **экземпляр простого объекта (простого объектного типа данных)** [simple entity (data type) instance];
- **граф подтипов/супертипов** (subtype/supertype graph);
- **лексема** (token);
- **значение** (value).

3.3 Другие термины и определения

В настоящем стандарте применены следующие термины с соответствующими определениями:

3.3.1 **связующее пространство** (binding extent): Множество связующих экземпляров, созданных из экземпляров, присутствующих в исходных пространствах объектных типов данных и пространствах образов.

3.3.2 **связующий экземпляр** (binding instance): Совокупность ссылок на экземпляры объектного типа данных и экземпляры типа данных образа, связанные с образом или отображением.

3.3.3 **пространство объектного типа данных** (entity data type extent): Совокупность экземпляров данного объектного типа данных.

3.3.4 **анализатор EXPRESS-X** (EXPRESS-X parser): Сервисная программа, способная выполнять синтаксический анализ детализированного описания, сформулированного на языке EXPRESS-X.

3.3.5 **программа отображения EXPRESS-X** (EXPRESS-X mapping engine): Программное средство, выполняющее отображение структурированной информации, основанное на детализированном описании, сформулированном на языке EXPRESS-X.

---

* Отменен. Действует ИСО/МЭК 10646:2012. Для однозначного соблюдения требований настоящего стандарта, выраженных в датированных ссылках, рекомендуется использовать только данный ссылочный стандарт.

3.3.6 **отображение** (map): Объявление взаимосвязи между данными, представляющими один или несколько исходных объектных типов данных или исходных типов данных образа, и данными, представляющими один или несколько целевых объектных типов данных.

3.3.7 **сетевое отображение** (network mapping): Отображение на множество целевых экземпляров объектов.

3.3.8 **квалифицированное связующее пространство** (qualified binding extent): Подмножество связующего пространства, состоящее из связующих экземпляров, соответствующих набору критериев выбора.

3.3.9 **критерий выбора** (selection criterion). Логическое выражение; критерий выбора соответствует требованиям только в том случае, если оценкой данного выражения является значение TRUE (ИСТИНА).

3.3.10 **исходный набор данных** (source data set): Совокупность экземпляров объектных типов данных, управляемых EXPRESS-схемой, являющаяся источником данных для отображения.

3.3.11 **исходное пространство** (source extent): Пространство образа или пространство объектных типов данных, предназначенное для создания связующего пространства.

3.3.12 **целевой набор данных** (target data set): Совокупность экземпляров объектов, созданных в результате отображения.

3.3.13 **образ** (view): Альтернативная организация информации в EXPRESS-схеме.

3.3.14 **тип данных образа** (view data type): Представление образа.

3.3.15 **экземпляр типа данных образа** (view data type instance): Именованный блок информации, созданный с помощью оценки образа.

3.3.16 **пространство образа** (view extent): Агрегированная структура экземпляров типов данных образа, содержащая все экземпляры, которые могут быть созданы из квалифицированного связующего пространства.

4 Основные принципы

4.1 Введение

Настоящий стандарт основан на изложенных в данном разделе основных принципах. Кроме того, к настоящему стандарту применимы понятия, описанные в ИСО 10303-11, раздел 5.

Язык EXPRESS-X обеспечивает спецификацию:

- различающихся образов данных, управляемых EXPRESS-схемой, с использованием объявлений образов (см. 9.3) в образе схемы (см. 9.5);
- отображения данных, управляемых одной или несколькими исходными EXPRESS-схемами, с использованием объявлений отображения (см. 9.4) в отображении схемы (см. 9.6).

Схема, представленная на языке EXPRESS-X, может содержать спецификации функций и процедур языка EXPRESS, для того чтобы поддержать определение образов и отображений.

Примечания

1 Соглашение об обозначениях, используемое в данной спецификации, должно контекстуализировать части предложения относительно обсуждения образов или отображений за счет включения в соответствующие места предложения терминов "(views [образы])" или "(maps [отображения])".

2 Соглашение об обозначениях, используемое в данной спецификации, заключается в том, что связующий экземпляр обозначается как упорядоченный набор имен экземпляров объектов или образов, разделенных запятыми (","), заключенный в угловые скобки ("<>"). Упорядочение имен экземпляров соответствует порядку появления исходного пространства в элементе языка FROM в объявлении образа или отображения.

Пример — Пусть заданы:

- объявление образа:

```
SCHEMA_VIEW my_person_org_achema_view;
REFERENCE FROM person_and_org_schema;
VIEW person_org;
  FROM p: person; o : organization; -- определяет порядок
  SELECT
    name : STRING := p.last_name;
    org : STRING := o.department_name;
END_VIEW;
END_SCHEMA_VIEW;
```

- исходная EXPRESS-схема:

```
SCHEMA person_and_org_schema;
ENTITY person;
  first_name : STRING;
  last_name : STRING;
END_ENTITY;
ENTITY organization;
  department_name : STRING;
END_ENTITY;
END_SCHEMA;
```

- данные, закодированные в соответствии с ИСО 10303-21 [2]:

```
#1=PERSON('James','Smith');
#2=PERSON('Fredrick','Jones');
#31=ORGANIZATION('Engineering');
#32=ORGANIZATION('Sales');
```

тогда связующие экземпляры для заданных образа и данных могут быть записаны в представленном ниже виде. Понятие связующих экземпляров определено в других разделах настоящего стандарта и не является необходимым для понимания данного примера. Однако необходимо отметить, что первый элемент каждого связующего экземпляра берется из пространства объекта `person`***, а второй элемент — из пространства объекта*** `organization`***. Данный порядок соответствует порядку появления объектов*** `person` ***и*** `organization` ***в элементе языка*** `FROM` ***данного образа:***

`{<#1,#31>,<#1,#32>,<#2,#31>,<#2,#32>}`.

4.2 Основные принципы модели исполнения

4.2.1 Введение

Настоящая спецификация определяет язык и модель исполнения. Модель исполнения состоит из двух фаз — процесса связывания и процесса реализации. При оценке образов и отображений используется общий процесс связывания, но их процессы реализации различаются.

4.2.2 Процесс связывания

Связующей средой является среда, в которой переменным присваиваются значения. Связующий экземпляр представляет собой структуру, которая связывает переменные, объявленные в элементе языка FROM из объявления образа или отображения. Элемент языка FROM ссылается на исходные пространства объектов и пространства образов. Границы значений берутся из этих исходных пространств. Каждый связующий экземпляр входит в набор, определяемый как декартово произведение ссылочных исходных пространств. Рассчитанный таким образом набор связующих экземпляров является связующим пространством для данного образа или отображения при заданных исходных пространствах. Связки переменных в связующем экземпляре образуют среду для оценки тела образа или отображения в процессе реализации, когда ссылочные данные в связующем экземпляре соотносятся со структурами, создаваемыми в целевой совокупности. Таким образом, каждый связующий экземпляр соответствует экземпляру типа данных образа (образов) либо экземплярам целевого объектного типа данных (отображениям) в целевой совокупности.

Исходные пространства отображений и образов должны быть пространствами объектных типов данных или пространствами образов.

Зацикливание между ссылками и исходными пространствами запрещено.

Примеры

1 В процессе связывания, относящемся к образу, данным и схеме и определенном в примере из 4.1, вычисляется связующее пространство объекта `person_org` `{<#1,#31>,<#1,#32>, <#2,#31>,<#2,#32>}`***. Данное пространство представлено ниже в табличной форме:***

Связующий экземпляр	person			organization	
	#	first_name	last_name	#	department_name
<#1,#31>	#1	'James'	'Smith'	#31	'Engineering'
<#1,#32>	#1	'James'	'Smith'	#32	'Sales'
<#2,#31>	#2	'Fredrick'	'Jones'	#31	'Engineering'
<#2,#32>	#2	'Fredrick'	'Jones'	#32	'Sales'

2 Приведенный ниже образ схемы invalid *является примером неправильного образа схемы, так как он содержит цикл ссылок (образ* a *ссылается на образ* b*, который ссылается на образ* a*):*

```
SCHEMA_VIEW invalid;
VIEW a;
  FROM some_b : b;
  attr1 : INTEGER := some_b.attr2 + 2;
END_VIEW;
VIEW b;
  FROM some_a : a;
  attr2 : INTEGER := some_a.attr1 * 3;
END_VIEW;
END_SCHEMA_VIEW;
```

4.2.3 Процесс реализации

Связующей средой является среда, в которой переменным присваиваются значения, используемые во время процесса реализации. Каждый связующий экземпляр содержит набор значений, которые должны быть использованы в качестве границ переменных. Процесс реализации образа является процессом оценки тела образа (см. 9.3.2) для каждого связующего экземпляра в связующем пространстве. Порядок оценки связующих экземпляров не определен.

Примеры

1 Процесс связывания, относящийся к объявлению схемы, данных и образа из примера в 4.1, дает в результате связующее пространство объекта person_org: {<#1,#31>,<#1,#32>, <#2,#31>,<#2,#32>}*. Объявление образа и данные, использованные в примере из 4.1, здесь приведены еще раз:*

```
VIEW person_org;
  FROM p: person; o : organization; -- определяет порядок
  SELECT
    name : STRING := p.last_name;
    org : STRING := o.department_name;
END_VIEW;
```

```
#1=PERSON('James','Smith');
#2=PERSON('Fredrick','Jones');
#31=ORGANIZATION('Engineering');
#32=ORGANIZATION('Sales');
```

Связующий экземпляр <#1,#31> *соответствует заданию экземпляра объектного типа данных* #1 *переменной* p *и объектного типа данных* #31 *переменной* o*. Оценка тела образа в данном связующем пространстве дает в результате экземпляр типа данных образа со значением* 'Smith' *для атрибута* name *и значением* 'Engineering' *для атрибута* org*. Экземпляры типа данных образа могут быть закодированы так же, как если бы они были экземплярами объектного типа данных, с помощью кодирования, определенного в ИСО 10303-21 [2]. Пространство образа для данного примера выглядит следующим образом:*

```
#100=PERSON_ORG('Smith', 'Engineering');  /* <#1,#31> */
#101=PERSON_ORG('Smith', 'Sales');        /* <#1,#32> */
#102=PERSON_ORG('Jones', 'Engineering');  /* <#2,#31> */
#103=PERSON_ORG('Jones', 'Sales');        /* <#2,#32> */
```

2 Целевая EXPRESS-схема и отображение схемы со структурой, подобной структуре образа схемы, использованного в предыдущем примере, могут быть определены следующим образом:

```
SCHEMA similar_target;
ENTITY person_org;
 name : STRING;
 org : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP similar;
REFERENCE FROM person_and_org_schema AS SOURCE;
REFERENCE FROM similar_target AS TARGET;
MAP person_org_map AS
  po : person_org;
 FROM
  p : person;
```

```
  o : organization;
 SELECT
  po.name := p.last_name;
  po.org := o.department_name;
END_MAP;
END_SCHEMA_MAP;
```

Оценка данных из предыдущего примера дает в результате следующие экземпляры объектного типа данных, управляемого схемой `similar_target`:

```
#100=PERSON_ORG('Smith','Engineering');        /* <#1,#31> */
#101=PERSON_ORG('Smith','Sales');              /* <#1,#32> */
#102=PERSON_ORG('Jones','Engineering');        /* <#2,#31> */
#103=PERSON_ORG('Jones','Sales');              /* <#2,#32> */
```

Если выражение в правой части задания тела объявления отображения (образа) состоит только из ссылок на атрибуты (.) от исходного объекта и ссылочный атрибут задан в явном виде, то отображение данного атрибута целевого объекта (атрибута образа) будет двунаправленным.

3 В приведенном ниже фрагменте, является ли атрибут `po.dept_number` ***двунаправленным или нет, зависит от сущности функции*** `dept_func`:

```
MAP person_org_map AS
 po : person_org;
FROM
 p: person;
 o : organization;
SELECT
 po.name := p.last name;           -- двунаправленный
 po.org : = o.department name;     -- двунаправленный
 po.industry_code := o.owning_enterprise.industry.code_num; -- двунаправленный
 po.dept number := dept_func(o.department_name); -- возможно, не двунаправленный
END_MAP;
```

В данном подразделе определены только основные аспекты модели исполнения. Детали процесса связывания описаны в 9.2. Детали процесса реализации для образов описаны в 9.3, а для отображений — в 9.4.

4.3 Среда исполнения

Язык EXPRESS-X не описывает среду исполнения. В частности, язык EXPRESS-X не определяет:

- как ссылки соотносятся с именами;
- как задаются наборы входных и выходных данных;
- как осуществляется отображение для экземпляров, которые не соответствуют EXPRESS-схеме.

При оценке образа создается пространство образа. При оценке отображения могут быть созданы экземпляры объектов в целевом наборе данных. Язык EXPRESS-X не определяет, какой эффект модификация исходных данных может оказать на пространства образов или целевые наборы данных после первоначального отображения.

5 Требования соответствия

5.1 Классы соответствия языка EXPRESS-X

5.1.1 Введение

Класс соответствия реализации анализатора или программы отображения EXPRESS-X показывает, какую часть языка поддерживает данная реализация. Объявления классифицируются с помощью подмножеств языка, как показано в таблице 1.

Таблица 1 — Объявления и подмножества языка EXPRESS-X

Объявление	Подмножество 1	Подмножество 2
Объявление образа	+	–
Объявление отображения	–	+

Окончание таблицы 1

Объявление	Подмножество 1	Подмножество 2
Объявление зависимого отображения	–	+
Объявление константы	+	+
Объявление функции	+	+
Объявление процедуры	+	+
Объявление правила	+	–

Разработчик анализатора или программы отображения EXPRESS-X должен установить любые ограничения, которые данная реализация накладывает на число и длину идентификаторов, на диапазон обрабатываемых чисел и на максимальную точность представления действительных чисел. Данные ограничения должны быть задокументированы для тестирования на соответствие.

5.1.2 Классы соответствия анализатора EXPRESS-X

Реализация анализатора EXPRESS-X должна обеспечивать синтаксический анализ любых формальных спецификаций, написанных на языке EXPRESS-X, которые согласуются с классом соответствия, присвоенным данной реализации. Считают, что анализатор EXPRESS-X соответствует конкретному уровню проверки, определенному в 5.1.4, если он может выполнить все проверки формальной спецификации, написанной на языке EXPRESS-X, необходимые для данного уровня (и всех нижележащих уровней).

Анализатор EXPRESS-X, относящийся к классу соответствия 1, должен анализировать все объявления из подмножества языка 1 (см. таблицу 1).

Анализатор EXPRESS-X, относящийся к классу соответствия 2, должен анализировать все объявления из подмножества языка 2 (см. таблицу 1).

Анализатор EXPRESS-X, относящийся к классу соответствия 3, должен анализировать все объявления, определенные в настоящем стандарте.

5.1.3 Классы соответствия программы отображения EXPRESS-X

Реализация программы отображения EXPRESS-X должна обеспечивать отображение любых формальных спецификаций, написанных на языке EXPRESS-X, которые согласуются с классом соответствия, присвоенным данной реализации. Отображение осуществляется по отношению к одному или нескольким наборам исходных данных; определение того, как эти наборы данных становятся доступными для программы отображения, находится вне области применения настоящего стандарта.

Программа отображения EXPRESS-X, относящаяся к классу соответствия 1, должна обеспечивать отображение всех объявлений из подмножества языка 1 (см. таблицу 1).

Программа отображения EXPRESS-X, относящаяся к классу соответствия 2, должна обеспечивать отображение всех объявлений из подмножества языка 2 (см. таблицу 1).

Программа отображения EXPRESS-X, относящаяся к классу соответствия 3, должна обеспечивать отображение всех объявлений, определенных в настоящем стандарте.

5.1.4 Проверка совместимости анализаторов EXPRESS-X

5.1.4.1 Введение

Формальная спецификация, написанная на языке EXPRESS-X, должна быть совместима с заданным уровнем проверки. Формальная спецификация совместима с заданным уровнем в том случае, когда все проверки, определенные для данного уровня, а также для всех нижележащих уровней, могут быть выполнены для данной спецификации.

5.1.4.2 Уровень 1: проверка ссылок

Данный уровень предусматривает проверку формальной спецификации на ее синтаксическую и ссылочную правильность. Формальная спецификация считается синтаксически правильной в том случае, если она соответствует синтаксису, сгенерированному при расширении основного синтаксического правила, представленного в приложении В. Формальная спецификация считается ссылочно правильной в том случае, если все ссылки на элементы языка EXPRESS-X соответствуют правилам области действия и области видимости, определенным в разделе 13.

5.1.4.3 Уровень 2: проверка типов данных

Данный уровень включает проверки уровня 1 и проверку формальной спецификации на ее совместимость по следующим позициям:

- выражения должны подчиняться правилам, определенным в разделе 10 и в ИСО 10303-11, раздел 12;

- назначения должны подчиняться правилам, определенным в ИСО 10303, подраздел 13.3.

5.1.4.4 Уровень 3: проверка значений

Данный уровень включает проверки уровня 2 и проверку формальной спецификации на ее совместимость с выражениями вида «А должно быть больше, чем В», определенным в ИСО 10303-11, разделы 7—14. При этом данные выражения ограничены случаями, когда значения А и В могут быть получены из литералов и/или констант.

5.1.4.5 Уровень 4: полная проверка

Данный уровень включает проверку формальной спецификации на ее совместимость со всеми требованиями, установленными в настоящем стандарте.

6 Синтаксис спецификации языка

В данном разделе определена нотация, используемая для представления синтаксиса языка EXPRESS-X.

Полный синтаксис языка EXPRESS-X приведен в приложении В. Части этих синтаксических правил воспроизведены в различных разделах настоящего стандарта для иллюстрации синтаксиса конкретных операторов. Эти части не всегда полны. Поэтому иногда необходимо руководствоваться приложением В в отношении недостающих в данном примере правил. Части синтаксических правил в тексте настоящего стандарта представлены в рамках. Каждое синтаксическое правило внутри рамки обозначено слева уникальным номером для использования его в перекрестных ссылках в других синтаксических правилах.

Синтаксис языка EXPRESS-X определен как производная от синтаксической нотации Вирта (СНВ) [1].

Соглашения об обозначениях и самоопределенная СНВ приведены ниже.

```
syntax          = { production } .
production      = identifier '=' expression '.' .
expression      = term { '|' term } .
term            = factor { factor } .
factor          = identifier | literal | group | option | repetition .
identifier      = character { character } .
literal         = '''' character { character } '''' .
group           = '(' expression ')' .
option          = '[' expression ']' .
repetition      = '{' expression '}' .
```

Знак равенства '=' обозначает порождающее правило. Элемент слева от знака равенства определяется как комбинация элементов, расположенных справа от него. Любые пробелы между элементами правой части не имеют значения, если только они не входят в состав литерала. В конце порождающего правила ставится точка '.'.

Использование идентификатора в любом элементе обозначает нетерминальный символ, который присутствует в левой части другого порождающего правила. Идентификатор состоит из букв, цифр и символа подчеркивания. Ключевые слова языка представлены порождающими правилами, идентификаторы которых состоят только из прописных букв.

Литерал используется для обозначения терминального символа, который не может быть раскрыт в дальнейшем. Литерал представляется последовательностью символов, заключенной в апострофы. Чтобы апостроф был включен в литерал, он должен быть записан дважды, т.е. ''''.

Семантика разных видов скобок определена следующим образом:

- фигурные скобки '{ }' обозначают ни одного или несколько повторений;

- квадратные скобки '[ ]' обозначают необязательные параметры;

- круглые скобки '( )' обозначают, что группа порождающих правил, заключенная в круглые скобки, должна использоваться как единое порождающее правило;

- вертикальная линия '|' обозначает, что в выражении должен использоваться только один из элементов, разделенных вертикальными линиями.

Следующая нотация используется для представления полных наборов символов и некоторых специальных символов, которые трудно визуально отобразить:

- \a — представляет любой символ из ИСО/МЭК 10646-1;
- \r — представляет символ новой строки (newline), роль которого зависит от системы (см. пункт 7.1.5.2 ИСО 10303-11).

7 Основные элементы языка

7.1 Введение

В данном разделе определены основные элементы, из которых формируется спецификация отображения EXPRESS-X: набор символов, комментарии, знаки, зарезервированные слова, идентификаторы и литералы.

Элементы языка EXPRESS-X совпадают с элементами языка EXPRESS, определенными в ИСО 10303-11, раздел 7, с некоторыми исключениями, отмеченными далее.

7.2 Зарезервированные слова

Зарезервированными словами языка EXPRESS-X являются ключевые слова и имена встроенных констант, функций и процедур. Все зарезервированные слова языка EXPRESS (см. ИСО 10303-11) являются зарезервированными словами языка EXPRESS-X. Зарезервированные слова не должны использоваться в качестве идентификаторов. Дополнительные зарезервированные слова языка EXPRESS-X определены в таблице 2.

Таблица 2 — Дополнительные ключевые слова языка EXPRESS-X

DEPENDENT_MAP	EACH	ELSIF	END_DEPENDENT_MAP
END_MAP	END_SCHEMA_MAP	END_SCHEMA_VIEW	END_VIEW
EXTENT	IDENTIFIED_BY	INDEXING	MAP
ORDERED_BY	PARTITION	SCHEMA_MAP	SCHEMA_VIEW
SOURCE	TARGET	VIEW	

Примечание — В том случае, если правильный идентификатор языка EXPRESS совпадает с зарезервированным словом языка EXPRESS-X, использующие данный идентификатор схемы могут быть отображены с помощью переименования конфликтующего идентификатора с использованием элемента языка REFERENCE (см. 13.2).

8 Типы данных

8.1 Введение

Определенный в данном разделе тип данных вместе с типами данных, определенными в языке EXPRESS (ИСО 10303-11, раздел 8), составляют часть языка EXPRESS-X.

Каждый атрибут образа (см. 9.3.2) имеет связанный с ним тип данных.

8.2 Тип данных образа

Типы данных образа устанавливаются с помощью объявлений образа (см. 9.3). Типу данных образа присваивается идентификатор при определении отображения схемы или образа схемы. Ссылки на тип данных образа осуществляются с помощью данного идентификатора.

Синтаксис:

```
230 view_reference = ? ( schema_map_ref | schema_view_ref ) '.' ] view_ref .
```

Правила и ограничения:

а) Элемент языка view_reference должен представлять собой ссылку на образ, видимый в текущей области действия.

b) Элемент языка view_reference не должен ссылаться на зависимый образ (см. 9.3.5).

Пример — Следующее объявление определяет тип данных образа с именем circle:

```
VIEW circle;
 FROM e : ellipse;
 WHERE (e.major_axis = e.minor_axis);
 SELECT
  radius : REAL := e.minor_axis;
  centre : point := e.centre;
END_VIEW;
```

9 Объявления

9.1 Введение

В данном разделе определены объявления, доступные в языке EXPRESS-X. Объявление в языке EXPRESS-X создает новый элемент языка и связывает с ним некоторый идентификатор. На элемент языка EXPRESS-X можно ссылаться в любом месте с помощью данного идентификатора.

В языке EXPRESS-X определены следующие объявления:

- образа;
- отображения;
- зависимого отображения;
- образа схемы;
- отображения схемы.

Кроме того, спецификация на языке EXPRESS-X может содержать следующие объявления, определенные в ИСО 10303-11:

- константы;
- функции;
- процедуры;
- правила.

9.2 Связывание

9.2.1 Введение

Связующее пространство представляет собой совокупность связующих экземпляров, созданных из экземпляров, существующих в пространствах исходных объектных типов данных и пространствах образов. Связующее пространство рассчитывается как декартово произведение пространств, на которые имеются ссылки в элементе языка FROM из объявления образа или отображения.

Квалифицированное связующее пространство является подмножеством связующего пространства, состоящим только из тех связующих экземпляров, для которых элемент языка WHERE возвращает значение TRUE при связывании их переменных со значениями в связующем экземпляре.

Примечание — Положения, определенные в 9.2, применяются к объявлениям отображений и объявлениям образов. Положения, применяемые только к объявлениям образов, определены в 9.3. Положения, применяемые только к объявлениям отображений, определены в 9.4.

9.2.2 Связующее пространство

Элемент языка FROM определяет элементы связующих экземпляров в связующем пространстве. Элемент языка FROM включает один или несколько параметров. Каждый исходный параметр связывает идентификатор с пространством.

Синтаксис:

```
228 view_decl = ( root_view_decl | dependent_view_decl   subtype view_decl
    ).
136 map_decl = MAP map_id AS target_parameter ';' { target_parameter ';' } (
    map_subtype_of_clause subtype_binding_header map_decl_body ) | (
    binding_header map_decl_body { binding_header map_decl_body } )
    END_MAP ';' .
 47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
    local_decl ] [ where_clause ] [ identified_by_clause ]
    ordered_by_clause ' .
 99 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
 83 extent_reference = source_entity_reference | view_reference .
```

Правила и ограничения

Идентификаторы source_parameter_id должны быть уникальными в области действия объявления отображения или образа.

Связующее пространство вычисляют как декартово произведение экземпляров пространств, на которые имеются ссылки в элементе языка FROM.

Пример — Связующее пространство строится над совокупностями объектов item ***и*** person.

```
SCHEMA source_schema;     -- EXPRESS-схема
ENTITY item;
 item_number : INTEGER;
 approved_by : STRING;
END_ENTITY;
ENTITY person;
 name : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA_VIEW example;
REFERENCE FROM source_schema;
VIEW items and persons;
 FROM i : item; p : person;
 SELECT
  item_number : INTEGER := i.item number;
  responsible : STRING := p.name;
END_VIEW;
END_SCHEMA_VIEW;
```

Пусть задана следующая совокупность (объекты представлены в соответствии с ИСО 10303-21):

```
#1=ITEM(123,'Smith');
#2=ITEM(234,'Smith');
#33=PERSON('Jones');
#44=PERSON('Smith');
```

тогда соответствующее связующее пространство будет выглядеть следующим образом: {<#1,#33>,<#1,#44>,<#2,#33>,<#2,#44>}.

9.2.3 Квалификация связующего пространства

Элемент языка WHERE определяет критерии выбора связующих экземпляров в связующем пространстве. Элемент языка WHERE вместе с исходными пространствами, определенными в элементе языка FROM, определяют квалифицированное связующее пространство.

Связующей средой является среда, в которой переменным присваиваются значения. Исходные параметры элемента языка FROM являются границами связующего экземпляра. Выражения для правил области действия элемента языка WHERE вычисляют в процессе данного связывания. Связующий экземпляр из связующего пространства включается в квалифицированное связующее пространство в том случае, если вычисленным значением всех выражений для правил области действия элемента языка WHERE является TRUE.

Синтаксис элемента языка WHERE определен в пункте 9.2.2.2 ИСО 10303-11.

Пример — Квалифицированное связующее пространство включает те пары объектов item ***и*** person ***из связующего пространства, для которых значением атрибута*** person.name ***является*** 'Smith' ***или*** 'Jones' ***и значением атрибута*** item.approved_by ***является также*** 'Smith' ***или*** 'Jones'.

```
SCHEMA_VIEW example;
REFERENCE FROM source_schema;
VIEW items and persons;
 FROM i : item; p : person;
 WHERE (p.name = 'Smith') OR (p.name = 'Jones');
    (i.approved by = p.name);
SELECT
 name : STRING := p.name;
END_VIEW;
END_SCHEMA_VIEW;
```

Квалифицированное связующее пространство, соответствующее данным из примера в 9.2.2, будет выглядеть следующим образом:
{<#1,#44>,<#2,#44>}.

9.2.4 Идентификация экземпляров образа и целевых экземпляров

Связующий экземпляр отображения или образа, не содержащий элемент языка identified by clause, идентифицируют значениями (экземплярами объектного типа данных), которые он получает из пространств, на которые имеются ссылки в элементе языка FROM. Связующий экземпляр отображения или образа, содержащий элемент языка identified by clause, идентифицируют значением (значениями) элемента языка expression в соответствии с синтаксическим правилом 108. Эти схемы идентификации используют при вызове образа (см. 10.2) и вызове отображения (см. 10.3).

Элемент языка identified by clause определяет отношение эквивалентности между экземплярами в связующем пространстве.

Синтаксис:
```
107 identified_by_clause = IDENTIFIED_BY id_parameter ';' {id_parameter
    ';'} .
108 id_parameter = [id_parameter_id ':'] expression .
109 id_parameter_id = simple_id .
 90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
```

Правила и ограничения

При использовании в объявлении отображения элемента языка expression в элементе id parameter синтаксические конструкции языка не должны ссылаться на экземпляры целевых объектов отображения или на любые их атрибуты при любом уровне косвенности.

Два связующих экземпляра относятся к одному классу эквивалентности, если для каждого выражения элемента языка identified by clause вычисление значения данного выражения в контексте этих экземпляров дает в результате то, что данные экземпляры равны (пункт 12.2.2 ИСО 10303-11). В процессе реализации создается один экземпляр образа (образов) или целевая сеть (отображения) для каждого класса эквивалентности.

***Пример — Данный пример иллюстрирует использование элемента языка* identified_by.**

```
SCHEMA_VIEW example;
REFERENCE FROM some_schema;
VIEW department;
 FROM e : employee;
 IDENTIFIED_BY e.department_name;
 SELECT
  name : STRING := e.department_name;
END_VIEW;
END_SCHEMA_VIEW;

SCHEMA some_schema;
ENTITY employee;
 name : STRING;
 department name : STRING;
END_ENTITY;
END_SCHEMA;

#1=EMPLOYEE('Jones','Engineering');
#2=EMPLOYEE('Smith','Sales');
#3=EMPLOYEE('Doe','Engineering');
```

Для представленных в данном примере образе и данных существуют два класса эквивалентности:* {<#1>,<#3>} *и* {<#2>}, *соответствующие связующим экземплярам с* e.department_name = 'Engineering' *и* e.department_name = 'Sales' *соответственно.

9.2.5 Классы эквивалентности и процесс реализации

Атрибуты образа (см. 9.3.2) и атрибуты целевого объекта (см. 9.4.2) представляют характеристики соответствующих образа (объявление образа) и объектов целевой сети (объявление отображения). Эти

атрибуты получают значения с помощью вычисления соответствующего выражения, представленного элементом языка expression (синтаксическое правило 224). Вычисление выражений производится в контексте связующего экземпляра из квалифицированного связующего пространства.

Синтаксис:

```
224 view_attribute_decl = view_attribute_id ':' [ OPTIONAL ] [
    source_schema_ref '.' ] base_type ':=' expression ';' .
234 map_attribute_declaration = [ target_parameter_ref [ index_qualifier ]
    [ group_qualifier ] '.' ] attribute_ref [ index_qualifier ] ':='
    expression ';' .
```

Если класс эквивалентности, определенный элементом языка identified by class, содержит более одного квалифицированного связующего экземпляра, то значение элемента expression вычисляется следующим образом:

- если существуют связующие экземпляры, для которых вычисление значения элемента expression (синтаксическое правило 224) не приводит к неопределенному результату, и если все такие значения, не являющиеся неопределенными, одинаковы (равенство экземпляров) или если существует только одно такое значение, то данное значение присваивается данному атрибуту;
- если для двух или более связующих экземпляров вычисление значения элемента expression дает не являющиеся неопределенными неравные результаты или если все вычисления производят неопределенные значения, то данному атрибуту присваивается неопределенное значение.

Пример — Данный пример иллюстрирует задание значений, при котором класс эквивалентности содержит несколько квалифицированных связующих экземпляров. Объявление отображения описано в 9.4.

```
(* Исходная схема *)
SCHEMA src;
ENTITY employee;
 name: STRING;
 manager: STRING;
 dept: STRING;
END_ENTITY;
END_SCHEMA;
```

```
(* Целевая схема *)
SCHEMA tar;
ENTITY department;
 employee: STRING;
 manager: STRING;
 dept_name: STRING;
END_ENTITY;
END_SCHEMA;
```

```
(* Схема отображения *)
SCHEMA_MAP example;
REFERENCE FROM src AS SOURCE;
REFERENCE FROM tar AS TARGET;
MAP department_map AS d : department;
 FROM e : src.employee;
 IDENTIFIED_BY e.dept;
 SELECT
  d.employee := e.name;
  d.manager := e.manager;
  d.dept_name := e.dept;
END_MAP;
END_SCHEMA_MAP;
```

```
#1=EMPLOYEE('Smith','Jones','Marketing');
#2=EMPLOYEE('Doe','Jones','Marketing');
```

В приведенном фрагменте целевой набор данных содержит один экземпляр объекта, `#1=DEPARTMENT($,'Jones','Marketing')`. ***Атрибут*** `department.dept_name` ***является неопределенным, так как результатом вычисления выражения для данного атрибута являются два разных значения*** `('Smith' и 'Doe')`.

9.2.6 Упорядочение экземпляров образа и целевых экземпляров

Элемент языка ORDERED BY определяет упорядочение связующих экземпляров квалифицированного связующего пространства. Обращения к процедуре частичного связывания (см. 10.4) в соответствующем разделе образа или отображения дают в результате агрегированные структуры, упорядоченные в соответствии с элементом языка ORDERED BY, если он присутствует.

Элемент expression в элементе языка ORDERED BY должен вычислять значения одинакового сопоставимого по упорядоченности типа данных для всех связок в данном разделе. К сопоставимым по упорядоченности типам данных относятся NUMBER, BINARY, STRING и ENUMERATION, а также их конкретизации. Результирующее упорядочение в связующем пространстве должно быть таким, чтобы значением выражения (e < f) не было FALSE, где e и f представляют значения, полученные при вычислении выражения expression (см. синтаксическое правило 148) для любых двух последовательных элементов связующего пространства, а < — оператор сравнения значений языка EXPRESS (ИСО 10303-11, пункт 12.2.1) . Если определены дополнительные элементы языка expression, то для каждого последующего элемента expression описанный процесс применяется к результату вычисления значения предыдущего элемента expression.

Синтаксис:

```
 47 binding_header = [ PARTITION partitioned_id ';' ] [ from_clause ] {
    local_decl } [ where_clause ] [ identified_by_clause ] [
    ordered_by_clause ].
148 ordered_by_clause = ORDERED BY expression { ',' expression } ';' .
```

Правила и ограничения

a) Выражение expression из синтаксического правила 148 не должно давать в результате неопределенное значение для любых связок в данном разделе.

b) В дочерних разделах образов и отображений не должен быть задан элемент языка ORDERED BY. Дочерние разделы образов и отображений должны наследовать упорядочение от своих родительских разделов, которое определяется элементом языка ORDERED BY, если он там задан.

c) Совокупность разделов образов, образующая иерархию, должна содержать не более одного элемента языка ORDERED BY.

d) Совокупность разделов отображений, связанных элементом map_subtype_of_clause (см. синтаксическое правило 141), должна содержать не более одного элемента языка ORDERED BY.

9.3 Объявление образа

9.3.1 Введение

Объявление образа создает тип данных образа и объявляет идентификатор, чтобы на него ссылаться.

Пример — Следующее объявление образа создает тип данных образа **arm_person_role_in_organization.**

```
VIEW arm_person_role_in_organization;
FROM
 pao : person_and_organization;
 ccdpaoa : cc_design_person_and_organization_assignment;
WHERE ccdpaoa.assigned_person_and_organization :=: pao;
SELECT
 person : person := pao.the_person;
 org : organization := pao.the_organization;
 role : label := ccdpaoa.role.name;
END VIEW;
```

Синтаксис:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
    ) .
177 root_view_decl = VIEW view_id [ supertype_constraint ] ';'
    Binding_header SELECT view_attr_decl_stmt_list { binding_header SELECT
    view_attr_decl_stmt_list } END VIEW ';' .
 47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] {
    local_decl } [ where_clause ] [ identified_by_clause ] [
    ordered_by_clause ] .
 90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
198 source_parameter = source_parameter_id ':' extent_reference .
 83 extent_reference = source_entity_reference | view_reference .
```

9.3.2 Атрибуты образа

Атрибут типа данных образа представляет характеристику образа. Значение атрибута экземпляра образа получают в результате вычисления значения выражения, представленного элементом языка expression из синтаксического правила 224.

Имя атрибута образа (view_attribute_id из синтаксического правила 224) представляет роль, которую играет связанное с ним значение в контексте образа, в котором присутствует данный атрибут.

Синтаксис:

```
226 view attr decl stmt list = { view attribute decl } .
224 view attribute decl = view attribute id ':' [ OPTIONAL ] [
        source schema ref '.' ] base type ':=' expression ';' .
```

Правила и ограничения

a) Значение, полученное в результате вычисления выражения, представленного элементом языка expression из синтаксического правила 224, должно быть совместимо по присваиванию с атрибутом образа base_type.

b) Любое имя атрибута образа view_attribute_id, объявленное в объявлении образа, должно быть уникально в рамках данного объявления.

c) Элемент языка OPTIONAL показывает, что значение данного атрибута может быть неопределенным. Использование элемента языка OPTIONAL не влияет на модель исполнения.

9.3.3 Разделы образа

Раздел образа является подмножеством пространства образа. Пространство образа представляет собой объединение его разделов. Объявление образа состоит из одного или нескольких объявлений разделов, в каждое из которых входят свои собственные элементы языка FROM и WHERE.

Пример — В ИСО 10303-201 прикладной объект* ORGANIZATION *может быть отображен из объекта* PERSON, ORGANIZATION *или* PERSON_AND_ORGANIZATION. *Отображение образа данной схемы на образ* arm_organization *определено следующим образом:

```
VIEW arm_organization;
PARTITION a_single_person;
  FROM p : person;
    ...
PARTITION a_single_organization;
  FROM o: organization;
    ...
PARTITION a_person_in_an_organization;
  FROM po: person_and_organization;
    ...
END VIEW;
```

Синтаксис:

```
228 view decl = ( root view decl | dependent view decl | subtype view decl
    ) .
177 root view decl = VIEW view id [ supertype constraint ] ';'
    Binding header SELECT view attr decl stmt list { binding header SELECT
    view attr decl stmt list } END VIEW ';' .
 67 dependent view decl = VIEW view id ':' base type ';' binding header
    RETURN expression { binding header RETURN expression } END VIEW ';' .
206 subtype view decl = VIEW view id subtype declaration ';'
    subtype binding header SELECT view attr decl stmt list {
    subtype binding header SELECT view attr decl stmt list } END VIEW ';'
    .
203 subtype binding header = [ PARTITION partition id ';' ] where clause .
 47 binding header = [ PARTITION partition id ';' ] [ from clause ] [
    local decl ] [ where clause ] [ identified by clause ] [
    ordered by clause ] .
```

Правила и ограничения

a) Если в объявлении образа присутствует более одного раздела, то элемент языка `partition_id` должен быть задан для каждого раздела.

b) Значение элемента `partition_id` должно быть уникальным в рамках объявления образа.

c) Все разделы объявления образа должны определять одинаковые атрибуты (включая имена и типы данных).

d) Атрибуты объявления образа должны быть представлены в одинаковом порядке в каждом из его разделов.

9.3.4 Постоянные разделы

Раздел, в котором отсутствуют синтаксические элементы `FROM`, `WHERE` и `IDENTIFIED_BY`, называется постоянным разделом. Постоянный раздел представляет один экземпляр образа без привязки к исходным данным.

Пример — Данный пример иллюстрирует использование постоянных разделов:

```
VIEW person;
PARTITION mary;
  SELECT
    name : STRING := 'Mary';
    age : INTEGER := 22;
PARTITION john;
  SELECT
    name : STRING := 'John';
    age : INTEGER := 23;
END VIEW;
```

9.3.5 Зависимые образы

Зависимым образом является образ, в котором нет определения атрибутов. В разделах зависимого образа определен элемент языка `RETURN expression` (см. синтаксическое правило 67). Вычисление значения элемента языка expression не должно давать в результате значение типа `AGGREGATE`. Вычисленное значение должно быть сопоставимо по типу данных с элементом языка `base_type` (см. синтаксическое правило 67).

Синтаксис:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
    ) .
 67 dependent_view_decl = VIEW view_id ':' base_type ';' binding_header
    RETURN expression { binding_header RETURN expression } END VIEW ';' .
```

Если класс эквивалентности, определенный элементом языка `identified_by` зависимого образа, содержит более одного квалифицированного связующего экземпляра, то возвращаемое значение вычисляется следующим образом:

- если для каждой такой связки элемент языка `RETURN expression` (см. синтаксическое правило 67) вычисляет одинаковые значения, то возвращается данное значение;
- если для двух или более связок элемент языка `RETURN expression` (см. синтаксическое правило 67) вычисляет разные значения, то возвращается неопределенное значение.

Примеры

1 В данном примере определен подтип типа данных `car`, у экземпляров которого атрибут `colour` имеет значение `'red'`:

```
VIEW red_car : car;
 FROM rc : car;
 WHERE rc.colour ='red';
 RETURN rc;
END_VIEW;
```

2 В данном примере определено пространство, члены которого являются строками. Эти строки поступают из двух источников:

```
VIEW owner_name : STRING;
PARTITION one;
   FROM po : person;
```

```
   RETURN po.name;
PARTITION two;
   FROM or : organization;
   RETURN org.name;
END_VIEW;
```

9.3.6 Определение подтипов образов

Язык EXPRESS-X допускает определение образов как подтипов других образов, при этом тип данных подтипа образа является конкретизацией своего супертипа. Это устанавливает наследственную взаимосвязь между типами данных образа, при которой подтип наследует атрибуты и критерии выбора своего супертипа. Образ является подтипом другого образа, если в его объявлении присутствует элемент языка SUBTYPE. Пространство типа данных подтипа образа является подмножеством пространства его супертипа в соответствии с критерием выбора, определенным элементом языка WHERE в данном подтипе.

Подтип образа наследует атрибуты от своего супертипа образа (или образов). Наследование атрибутов должно соответствовать правилам и ограничениям наследования атрибутов, определенным в пункте 9.2.3.3 ИСО 10303-11, .

В объявлении подтипа образа могут быть заново определены атрибуты, присутствующие у одного из его супертипов. Новое определение атрибутов должно соответствовать правилам и ограничениям повторного объявления атрибутов, определенным в пункте 9.2.3.4 ИСО 10303-11.

При вычислении значения образа должен быть создан экземпляр образа в том случае, если удовлетворяется критерий выбора для самого общего супертипа. Экземпляр образа должен относиться к типу, соответствующему образу подтипа, если оценка всех критериев выбора для данного подтипа образа, а также всех его супертипов образа дает значение TRUE.

Синтаксис:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
    ) .
206 subtype_view_decl = VIEW view_id subtype_declaration ';'
    subtype_binding_header SELECT view_attr_decl_stmt_list {
    subtype_binding_header SELECT view_attr_decl_stmt_list } END_VIEW ';'
    .
205 subtype_declaration = SUBTYPE OF '(' view_ref { ',' view_ref } ')' .
203 subtype_binding_header = [ PARTITION partition_id ';' ] where_clause .
```

Правила и ограничения

a) Только в одном супертипе образа для данного подтипа образа должен быть определен элемент языка FROM.

b) Множество разделов подтипа образа должно быть подмножеством множества разделов его супертипа образа.

Примеры

1 В данном примере показано определение подтипа образа. Подтип образа **male** ***определяет дополнительное требование принадлежности к данному подтипу для экземпляров образа (атрибут*** **gender = 'M'***):***

```
VIEW person;
 FROM e:employee;
END_VIEW;

VIEW male SUBTYPE OF (person);
 WHERE e.gender = 'M';
...
END_VIEW;
```

2 В данном примере показано использование разделов и подтипов образа:

```
VIEW j;
PARTITION first;
 FROM s:three, t:four
 WHERE cond6;
```

```
…
PARTITION second;
 FROM r:four, q:five
 WHERE cond7;
…
END_VIEW;

VIEW k SUBTYPE OF (j);
 PARTITION second;
WHERE cond9;
…
END_VIEW;
```

Только тот подтип образа, для которого `k` ***является супертипом, может включать раздел*** `second`.

9.3.7 Ограничения супертипа

В объявлении образа могут быть определены ограничения супертипа (см. пункт 9.2.4 ИСО 10303-11). Удовлетворяется или нет ограничение супертипа не влияет ни на выполнение модели, ни на содержимое пространств образа.

Синтаксис:

```
228 view_decl = ( root_view_decl | dependent_view_decl | subtype_view_decl
    ) .
177 root_view_decl = VIEW view_id [ supertype_constraint ] ';'
    binding_header SELECT view_attr_decl_stmt_list { binding_header SELECT
    view_attr_decl_stmt_list } END_VIEW ';' .
207 supertype_constraint = abstract_supertype_declaration | supertype_rule
    .
 33 abstract_supertype_declaration = ABSTRACT SUPERTYPE [
    subtype_constraint ] .
205 subtype_declaration = SUBTYPE OF '(' view_ref { ',' view_ref } ')' .
204 subtype_constraint = OF '(' supertype_expression ')' .
208 supertype_expression = supertype_factor { ANDOR supertype_factor } .
209 supertype_factor = supertype_term { AND supertype_term } .
211 supertype_term = view_ref | one_of | '(' supertype_expression ')' .
147 one_of = ONEOF '(' supertype_expression { ','supertype_expression }
    ')' .
```

Пример

```
VIEW a ABSTRACT SUPERTYPE OF ONEOF(b ANDOR c, d);
…
END_VIEW;
```

Экземпляр образа `a` ***является допустимым в том случае, если у него есть по крайней мере два типа данных (*** `a` ***и какой-нибудь еще), что определено наличием ключевого слова*** `ABSTRACT`, ***а одним из возможных других типов данных может быть*** `d` ***или некоторая комбинация*** `b` ***и*** `c`, ***что определено наличием ключевого слова*** `ONEOF`.

9.4 Объявление отображения

9.4.1 Введение

Объявление отображения поддерживает задание соответствия между определениями объектных типов данных из двух или более EXPRESS-схем. Данное объявление поддерживает также отображение из множества определений исходных объектных типов данных на множество определений целевых объектных типов данных.

Синтаксис:

```
136 map_decl = MAP map_id AS target_parameter ',' { target_parameter ',' }
    ( map_subtype_of_clause subtype_binding_header map_decl_body ) | (
    binding_header map_decl_body { binding_header map_decl_body } )
    END_MAP ';' .
 47 binding_header = [ PARTITION partition_id ',' ] { from_clause } [
    local_decl ] { where_clause } [ identified_by_clause ] [
    ordered_by_clause ] .
203 subtype_binding_header = [ PARTITION partition_id ',' ] where_clause .
 90 from_clause = FROM source_parameter ',' { source_parameter ',' } .
198 source_parameter = source_parameter_id ':' extent_reference .
 83 extent_reference = source_entity_reference | view_reference .
137 map_decl_body = ( entity_instantiation_loop {
    entity_instantiation_loop } ) | map_project_clause | ( RETURN expression
     ';' ) .
214 target_parameter = target_parameter_id { ',' target_parameter_id } ':'
    [ AGGREGATE [ bound_spec ] OF ] target_entity_reference .
213 target_entity_reference = entity_reference { '&' entity_reference } .
```

Правила и ограничения

Элемент `map_id` из синтаксического правила 136 присваивает имя объявлению отображения.

Пример — В данном примере объект `pump` ***из исходного набора данных отображается на объекты*** `product` ***и*** `product_related_product_category`**:**

```
SCHEMA source_schema;
ENTITY pump;
 id, name : STRING;
END_ENTITY;
END_SCHEMA;
SCHEMA target_schema;
ENTITY product;
 id, name : STRING;
END_ENTITY;
ENTITY product_related_product_category;
 name : STRING;
 products : SET OF product;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP pump_mapping;

REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP network_for_pump AS
  pr : product;
  prpc : product_related_product_category;
FROM p : pump;
 SELECT

  pr.id := p.id;
  pr.name := p.name;
  prpc.name := 'pump';
  prpc.products := [ pr ];
END_MAP;
END_SCHEMA_MAP;
```

Необходимо отметить, что в данном примере для каждого созданного экземпляра объекта `product` ***существует точно один созданный экземпляр объекта*** `product_related_product_category`**.**

Исходные значения атрибутов вновь созданного экземпляра (экземпляров) являются неопределенными. Если такой атрибут не задан в теле отображения, то его значение остается неопределенным.

9.4.2 Вычисление значения тела отображения

Синтаксис:

```
136 map decl = MAP map id AS target parameter ,;' { target parameter ,;' } (
    map subtype of clause subtype binding header map decl body ) | (
    binding header map decl body { binding header map decl body } )
    END MAP ,;' .
137 map decl body = ( entity instantiation loop |
    entity instantiation loop } ) | map project clause | ( RETURN expression
     ,;' ) .
139 map project clause = SELECT { map attribute declaration } .
134 map attribute declaration = [ target parameter ref [ index qualifier ]
    ! group qualifier ] ,.' ] attribute ref [ index qualifier ] ,:='
    expression ,;' .
```

Синтаксические элементы `map attribute declaration` и определяющий его `map decl body` должны присваивать значения атрибутам экземпляров целевых объектов. Элемент `expression` должен вырабатывать значение, совместимое по присваиванию с атрибутом целевого объекта (см. подраздел 13.3 ИСО 10303-11).

Объект `map decl body`, определяющий элемент `RETURN`, должен вычислять значение выражения, указанного после ключевого слова `RETURN`. Результатом данного вычисления должна стать реализация экземпляров целевых объектов, совместимых по типу с объектными типами данных, определяемыми целевыми параметрами. Должны быть реализованы экземпляры объектов для всех целевых параметров.

9.4.3 Итерационный процесс для одного связующего экземпляра

9.4.3.1 Введение

При вычислении отображения могут быть созданы агрегированные структуры экземпляров целевых объектных типов данных. Начальное значение этих агрегированных структур является неопределенным.

Элементы языка `instantiation loop control` и `repeat control` обеспечивают следующие виды итераций:

- итерация над совокупностью экземпляров агрегированной структуры языка EXPRESS;
- итерация, наращивающая значение числовой переменной.

Синтаксис:

```
 77 entity instantiation loop = FOR instantiation loop control ';'
    map project clause .
139 map project clause = SELECT { map attribute declaration } .
119 instantiation loop control = instantiation foreach control |
    repeat control .
118 instantiation foreach control = EACH variable id IN expression { AND
    variable id IN expression } [ INDEXING variable id ] .
171 repeat control = [ increment control ] [ while control ] [
    until control ] .
```

Правила и ограничения

a) Элемент языка `map project clause` (см. синтаксическое правило 139) устанавливает локальную область действия, в которой все переменные цикла `variable id` (см. синтаксическое правило 118) определены неявным образом.

b) Типом данных переменной, неявно объявленной элементом `variable id`, расположенным перед ключевым словом `IN` (см. синтаксическое правило 118), является тип данных выражения.

c) Типом данных переменной, неявно объявленной элементом `variable id`, расположенным после ключевого слова `INDEXING` (см. синтаксическое правило 118), является тип данных `INTEGER`.

Элемент variable id, расположенный после ключевого слова INDEXING (см. синтаксическое правило 118), инициализируется значением 1 в начале первого итерационного цикла и наращивается на 1 в начале каждого последующего цикла.

9.4.3.2 Управление с помощью числового приращения

Синтаксический элемент repeat control обеспечивает итерационный процесс для одного связующего экземпляра с помощью оператора REPEAT языка EXPRESS (см. подраздел 13.9 ИСО 10303-11).

Пример — В данном примере показано использование синтаксического элемента **repeat_control** ***в целевой реализации на языке EXPRESS-X. Совокупность целевых экземпляров объекта*** **child** ***создается для каждого исходного объекта*** **parent**. ***Число созданных экземпляров определяет атрибут*** **number_of_children** ***объекта*** **parent**.

```
SCHEMA src;                          SCHEMA tar;
ENTITY parent;                       ENTITY parent;
number_of_children: INTEGER;         END_ENTITY;
END_ENTITY;                          ENTITY child;
END_SCHEMA;                             parent : parent;
                                     END_ENTITY;
                                     END_SCHEMA;

SCHEMA_MAP example;
  REFERENCE FROM src AS SOURCE;
  REFERENCE FROM tar AS TARGET;

MAP parent_map AS tp : tar.parent;
 FROM sp : src.parent;
 SELECT
END_MAP;

MAP children_map AS c : AGGREGATE OF child;
 FROM p : src.parent;
 FOR I := 1 TO p.number_of_children;
 SELECT
  c[i].parent := tp(p);
END_MAP;
END_SCHEMA_MAP;
```

9.4.3.3 Управление с помощью итераций на агрегированной структуре

При выполнении синтаксического элемента instantiation foreach control на каждом шаге итерации следующий элемент указанной агрегированной структуры связывается с переменной и, возможно, индекс, определяющий позицию данного элемента, связывается с переменной итератора. Область действия данных привязок переменных включает элемент map project clause.

Пример — В данном примере все версии одного объекта сгруппированы вместе в исходном множестве данных. В целевом множестве каждая версия объекта является экземпляром.

```
SCHEMA tar;
ENTITY item_version;
  item_id    : INTEGER;
  version_id : INTEGER;
END_ENTITY ;
END_SCHEMA;

SCHEMA src;
ENTITY item_with_versions;
  id   : INTEGER;
  id_of_versions : LIST OF INTEGER;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP mapping_schema;
REFERENCE FROM src AS SOURCE;
REFERENCE FROM tar AS TARGET;
MAP item_version_map AS
 iv : AGGREGATE OF item_version;
```

```
FROM
 iwv : item_with_versions;
FOR EACH version_iterator IN iwv.id_of_versions INDEXING i;
SELECT
  iv[i].item_id    := iwv.id;
  iv[i].version_id := version_iterator;
END_MAP;

END_SCHEMA_MAP;
```

Например, следующее исходное множество экземпляров:

```
#1 = ITEM_WITH_VERSIONS(1,(10,11,12));
```

создает следующее целевое множество экземпляров:

```
#1 = ITEM_VERSION(1,10);
#2 = ITEM_VERSION(1,11);
#3 = ITEM_VERSION(1,12);
```

В синтаксическом элементе `instantiation foreach control` может быть определено несколько выражений с помощью факультативного синтаксического элемента `AND` (см. синтаксическое правило 118). Итерации продолжаются до тех пор, пока по крайней мере одна исходная агрегированная структура не будет исчерпана. Элементу `variable id` исчерпанных агрегированных структур присваивается неопределенное значение.

9.4.4 Разделы отображения

Экземпляры объектного типа данных могут по-разному связываться с исходными данными. Для определения этих разных связей могут использоваться разделы отображения.

Если множество целевых объектов перечислено в заголовке объявления отображения, то разные подмножества данных объектов могут быть созданы в каждом подразделе.

Синтаксис:

```
136 map_decl = MAP map_id AS target_parameter ';' { target_parameter ';' }
    ( map_subtype_of_clause subtype_binding_header map_decl_body ) | (
    binding_header map_decl_body { binding_header map_decl_body } )
    END_MAP ';' .
 47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ] [
    local_decl ] [ where_clause ] [ identified_by_clause ] [
    ordered_by_clause ] .
```

Правила и ограничения

a) Если в объявлении отображения существует более одного раздела, то для каждого раздела должен быть задан элемент `partition_id`.

b) Каждый элемент `partition_id` в рамках объявления отображения должен быть уникальным.

c) Для каждого целевого объектного типа данных, на который имеется ссылка в заголовке отображения, по крайней мере в одном из разделов объявления отображения должны быть созданы экземпляры.

Пример — Данный пример иллюстрирует, как разные исходные объектные типы данных могут быть отображены на один целевой объектный тип данных с помощью объявления отображения, содержащего разделы.

```
(* Исходная схема *)
SCHEMA src;
ENTITY student;
     name: STRING;
END_ENTITY;
ENTITY employee;
  name: STRING;
END_ENTITY;
END_SCHEMA;
```

```
(* Целевая схема *)
SCHEMA tar;
ENTITY person;
  name: STRING;
END_ENTITY;
END_SCHEMA;
```

```
(* Схема отображения *)
SCHEMA_MAP example;
REFERENCE FROM src AS SOURCE;
REFERENCE FROM tar AS TARGET;

MAP student_employee_to_person AS p : tar.person;
PARTITION student;
FROM s : src.student;
SELECT
  p.name := s.name;
PARTITION employee;
FROM e : src.employee;
SELECT
  p.name := e.name;
END_MAP;
END_SCHEMA_MAP;
```

9.4.5 Отображение на тип данных и его подтипы

Объявления отображений могут быть представлены в виде иерархии типов и подтипов данных. В объявлениях отображений подтипов данных может быть расширено множество экземпляров объектов, созданных отображением их супертипа. Определение задания значений целевых атрибутов, объявленное в отображении супертипа, наследуется отображениями его подтипов. Таким образом, шаблон наличия наследования в целевой схеме может быть дублирован в объявлениях отображений.

Синтаксис:

```
136 map_decl = MAP map_id AS target_parameter ';' { target_parameter ';' }
    ( map_subtype_of_clause subtype_binding_header map_decl_body ) | (
    binding_header map_decl_body { binding_header map_decl_body } )
    END_MAP ';' .
141 map_subtype_of_clause = SUBTYPE OF '(' map_reference ')' ';' .
```

Наследование между объявлениями отображений должно соответствовать следующим правилам:

- объявление отображения подтипа должно быть выполнено в том случае, если вычисленные значения его правила `WHERE` и правил `WHERE` всех отображений его супертипов равны `TRUE`;
- объявление отображения подтипа наследует все целевые параметры от объявлений отображений его супертипов;
- в объявлении отображения подтипа может быть повторно объявлен тип данных любого целевого параметра из объявлений его супертипов. Тип данных повторно объявленного целевого параметра должен быть конкретизацией каждого типа данных целевого параметра, объявленного в отображениях его супертипов. Правила конкретизации определены в пункте 9.2.6 ИСО 10303-11;
- в объявлении отображения подтипа могут быть добавлены новые целевые параметры с помощью объявления дополнительного синтаксического элемента `target_parameter` с атрибутом `target_parameter_id`, которые должны отличаться от использованных в объявлениях отображений его супертипов;
- если в отображении супертипа объявлен целевой параметр типа данных `SELECT` (см. пункт 8.4.2 ИСО 10303-11), `GENERIC` (см. пункт 9.5.3.2 ИСО 10303-11) или объектного типа данных, объявленного как `ABSTRACT` (см. пункт 9.2.4.1 ИСО 10303-11), и не выполняется объявление отображения подтипа, которое повторно объявляет данный целевой параметр, то для данного целевого параметра не должны быть созданы экземпляры.

Расширяет ли отображение подтипа множество экземпляров объектов, созданных отображением его супертипа, или конкретизирует эти созданные экземпляры зависит от того, ссылается ли отображение подтипа на синтаксические элементы `target_parameter_id`, объявленные в отображении супертипа, или объявляет дополнительные синтаксические элементы `target_parameter_id`. В отображении подтипа может быть введен синтаксический элемент `target_parameter_id`, который не определен ни в одном из отображений супертипов. При этом создается новый целевой объект, имеющий тип данных, определенный целевым параметром.

В отображении подтипа может присутствовать ссылка на задание целевого атрибута, на задание которого имеется ссылка в одном из его супертипов (возможно, через несколько уровней наследования). При этом целевому атрибуту задается значение, соответствующее значению самого конкретизированного отображения, которому соответствуют критерии выбора, а также критерии выбора его супертипов.

Отображение подтипа должно иметь только одно отображение непосредственного супертипа.

Пример — Данный пример иллюстрирует задание атрибутов, объявленных в супертипах и подтипах с помощью отображений супертипов и подтипов. Исходные объекты относятся к одному типу данных `s_project`. ***Целевые объекты относятся к типу данных*** `t_project` ***и, возможно, к одному из его подтипов*** `in_house_project` ***или*** `external_project`. ***Синтаксический элемент*** `target_parameter_id`, ***представленный элементом*** `tp`, ***использованным в отображении супертипа (*** `project_map`***), используется снова в отображениях его подтипов (*** `in_house_map`, `ext_map`***), указывая на то, что соответствующий целевой объект конкретизирован в отображениях подтипов.***

```
SCHEMA source_schema;
ENTITY s_project;
  name : STRING;
  project_type : STRING;
  cost : INTEGER;
  price : INTEGER;
  vendor : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA target_schema;
ENTITY t_project
 SUPERTYPE OF (ONEOF (in_house_project, external_project));
  name : STRING;
  cost : INTEGER;
  management : STRING;
END_ENTITY;

ENTITY in_house_project
 SUBTYPE OF (t_project);
END_ENTITY;

ENTITY external_project
 SUBTYPE OF (t_project);
  price : INTEGER;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP example;
REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;
MAP project_map AS tp : target_schema.t_project;
 FROM p : source_schema.s_project;
 SELECT
     tp.name := p.name;
     tp.cost := p.cost;
END_MAP;

MAP in_house_map AS tp : target_schema.in_house_project;
 SUBTYPE OF (project_map);
 WHERE (p.project_type = 'in house');
 SELECT
  tp.management := IF (p.cost < 50000) THEN 'small accts'
                   ELSE 'large accts' END_IF;
END_MAP;

MAP ext_map AS tp : target_schema.external_project;
 SUBTYPE OF (project_map);
 WHERE (p.project_type = 'external');
```

```
 SELECT
  tp.price := p.price;
  tp.management := p.vendor;
END_MAP;
END_SCHEMA_MAP;
```

В отображении супертипа могут быть определены циклы создания экземпляров объектов. Отображение подтипа данного отображения супертипа должно наследовать эти циклы создания экземпляров. Тело цикла создания экземпляров может быть повторно определено. Соответствие между телами циклов в отображении супертипа и в отображении подтипа, в котором используются циклы создания экземпляров, реализуется с помощью использования идентичных индексных идентификаторов; в теле из отображения подтипа, наследующем цикл, должна использоваться ссылка на индексный идентификатор, идентичный индексному идентификатору, определенному в отображении супертипа.

Пример — Данный пример иллюстрирует наследование цикла создания экземпляров объектов.

```
SCHEMA source_schema;
ENTITY part;
  name : STRING;
  no_of_versions : INTEGER;
  is_assembly : BOOLEAN;
END_ENTITY;
END_SCHEMA;

SCHEMA target_schema;
ENTITY product;
  name : STRING;
END_ENTITY;

ENTITY product_version;
  version_id : INTEGER;
  of_product : product;
END_ENTITY;

ENTITY product_definition;
  name : STRING;
  of version : product_version;
END_ENTITY;

END_SCHEMA;

SCHEMA_MAP example;
REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP super_map AS
  pvw  : AGGREGATE OF product_definition;
  pver : AGGREGATE OF product_version;
  pro  : product;
 FROM prt : part;
 FOR i := 1 TO prt.no_of_versions;
 SELECT
  pver[i].version_id := i;
  pver[i].of_product := pro;
  pvw[i].of_version := pver[i];
  pvw[i].name := 'view of part ' + prt.name;
SELECT
  pro.name := 'part ' + prt.name;
END_MAP;

MAP sub_map AS
  pro : product;
 SUBTYPE OF (super_map);
```

```
 WHERE
  prt.is_assembly = TRUE;
 SELECT
  pvw[i].name := 'view of assembly ' + prt.name;
  pro.name := 'assembly ' + prt.name;
END_MAP;
END_SCHEMA_MAP;
```

9.4.6 Объявление в явном виде сложных объектных типов данных

Сложные объектные типы данных (см. пункт 3.3.1 ИСО 10303-11) могут быть объявлены в явном виде в заголовке отображения. Сложный объектный тип данных обозначается с помощью синтаксической конструкции, в которой приводится список составляющих сложного объектного типа данных, комбинация которых образует данный сложный объектный тип данных, разделенных символом '&'.

Список составляющих сложного объектного типа данных может быть представлен в произвольном порядке.

Синтаксис:

```
214  target_parameter = target_parameter_id { ,,' target_parameter_id } ,:'
     [ AGGREGATE [ bound_spec ] OF ] target_entity_reference .
213  target_entity_reference = entity_reference { ,&' entity_reference } .
 78  entity_reference = [ ( source_schema_ref | target_schema_ref |
     schema_ref ) ,.' ] entity_ref .
```

Правила и ограничения

a) Каждый элемент `entity_ref` должен быть ссылкой на определение объектного типа данных, видимого в текущей области действия.

b) Ссылочный сложный объектный тип данных должен описывать допустимую область действия в рамках целевой схемы (см. приложение В ИСО 10303-11).

c) Элемент `ENTITY_REFERENCE` должен присутствовать не менее одного раза в определении элемента `TARGET_ENTITY_REFERENCE`.

d) Для каждого элемента `ENTITY_REFERENCE`, объявленного в определении элемента `TARGET_ENTITY_REFERENCE`, не должен быть объявлен ни один из его супертипов.

Пример — Данный пример иллюстрирует использование синтаксических конструкций с символом '&' для определения сложных объектных типов данных.

```
SCHEMA source_schema;

ENTITY pump;
 id, name : STRING;
END_ENTITY;

END_SCHEMA;

SCHEMA target_schema;

ENTITY item
ABSTRACT SUPERTYPE OF (ONEOF (product, kitchen_appliance));
END_ENTITY;

ENTITY product SUBTYPE OF (item);
 id, name :STRING;
END_ENTITY;

ENTITY kitchen_appliance SUBYPE OF (item);
END_ENTITY;

ENTITY product_related_product_category;
 name : STRING;
 products : SET OF product;
```

```
END_ENTITY;

END_SCHEMA;

SCHEMA_MAP example;

REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP network_for_pump AS
 pr : product & kitchen_appliance;
 prpc : product_related_product_category;
FROM p : pump;
 SELECT
  pr.id := p.id;
  pr.name := p.name;
  prpc.name := 'pump';
  prpc.products := [ pr ];
END MAP;

END_SCHEMA_MAP;
```

9.4.7 Зависимое отображение

Зависимым отображением является отображение, которое выполняется только с помощью вызова отображения (см.10.3). У зависимого отображения исходные параметры должны быть представлены только простыми и/или объектными типами данных.

Синтаксис:

```
 66  dependent_map_decl = DEPENDENT MAP map_id AS target_parameter {
     target_parameter } [ map_subtype_of_clause ] dep_map_partition {
     dep_map_partition } END_DEPENDENT MAP ';' .
214  target_parameter = target_parameter_id { ',' target_parameter_id } ':'
     ( AGGREGATE [ bound_spec ] OF ) target_entity_reference .
 71  dep_map_partition = [ PARTITION partition_id ':' ] dep_map_decl_body .
 70  dep_map_decl_body = dep_binding_decl_map_project_clause .
 68  dep_binding_decl = dep_from_clause [ where_clause ] [
     ordered_by_clause ] .
 69  dep_from_clause = FROM dep_source_parameter ';' { dep_source_parameter
     ';' } .
 72  dep_source_parameter = source_parameter_id { ',' source_parameter_id }
     ':' ( simple_types | type_reference ) .
```

Правила и ограничения

a) Если существует более одного раздела, то для каждого раздела должен быть задан элемент `partition_id`.

b) Для каждого целевого объектного типа данных, на который имеется ссылка в заголовке зависимого отображения, по крайней мере в одном из разделов объявления отображения должны быть созданы его экземпляры.

c) Значения элементов `partition_id` должны быть уникальными в области действия объявления зависимого отображения.

d) Объявление зависимого отображения, содержащее элемент `map_subtype_of_clause`, должно содержать только один раздел.

Пример — Данный пример иллюстрирует использование зависимого отображения для создания целевых экземпляров объекта `organization`***, имеющих уникальные атрибуты*** `id`***. Вызов зависимого отображения обеспечивает то, что экземпляры объекта*** `organization` ***из целевой совокупности имеют уникальные атрибуты*** `id`***, так как вызовы отображения с идентичными аргументами возвращают такие же экземпляры целевых объектов. См. 10.3.***

```
SCHEMA source;
ENTITY named_organization;
  name : STRING;
END_ENTITY;
ENTITY id_organization;
  id : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA target_schema;
ENTITY organization;
 id : STRING;
UNIQUE
 url: id;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP example;

REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP unique_orgs_map AS org : organization;
 PARTITION a_org;
 FROM a : named_organization;
   RETURN org_map(a.name);
 PARTITION b_org;
 FROM b : id_organization;
   RETURN org_map(b.id);
 END_MAP;

DEPENDENT_MAP org_map AS org : organization;
 FROM id : STRING;
 SELECT
   org.id := id;
END_DEPENDENT_MAP;

END_SCHEMA_MAP;
```

9.5 Объявление образа схемы

Объявление образа схемы определяет общую область действия для совокупности взаимосвязанных объявлений. Образ схемы может содержать любые объявления из подмножества языка 1 (см. таблицу 1).

Порядок, в котором объявления присутствуют в объявлении образа схемы, не имеет значения.

Объявления из одного образа схемы или EXPRESS-схемы могут быть сделаны видимыми в рамках области действия другого образа схемы с помощью задания интерфейса, определенного в разделе 13.

Синтаксис:

```
189 schema_view_decl = SCHEMA VIEW schema_view_id ';' { reference_clause }
    [ constant_decl ] schema_view_body_element_list END_SCHEMA VIEW ';' .
167 reference_clause = REFERENCE FROM schema_ref_or_rename [ '(,
    resource_or_rename { ',' resource_or_rename } ')' ] [ AS ( SOURCE |
    TARGET ) ] ';' .
188 schema_view_body_element_list = schema_view_body_element {
    schema_view_body_element } .
187 schema_view_body_element = function_decl | procedure_decl | view_decl
    | rule_decl .
```

Правила и ограничения

Синтаксический элемент AS (SOURCE | TARGET) не должен использоваться в схеме образа схемы.

Пример — ap203_arm является именем образа схемы, который может содержать объявления, определяющие образ схемы config_control_design.

```
SCHEMA_VIEW ap203_arm;
REFERENCE FROM config_control_design;
VIEW part version ...
(* другие возможные объявления *)
END_SCHEMA_VIEW;
```

9.6 Объявление отображения схемы

Объявление отображения схемы определяет общее пространство для совокупности взаимосвязанных объявлений. Отображение схемы должно идентифицировать одну или несколько схем как исходные схемы и одну или несколько схем как целевые схемы (см. 13.2). Отображение схемы не ограничено объявлениями из подмножеств языка EXPRESS-X (см. таблицу 1).

Порядок, в котором объявления присутствуют в объявлении отображения схемы, не имеет значения.

Объявления из одного отображения схемы могут быть сделаны видимыми в рамках области действия другого отображения схемы с помощью задания интерфейса, определенного в разделе 13.

Синтаксис:

```
184 schema_map_decl = SCHEMA_MAP schema_map_id ';' reference_clause {
    reference_clause } [ constant_decl ] schema_map_body_element_list
    END_SCHEMA_MAP ';' .
182 schema_map_body_element = function_decl | procedure_decl | view_decl |
    map_decl | dependent_map_decl | rule_decl .
```

Правила и ограничения

Отображение схемы должно включать в явном виде или через ссылку с использованием элемента языка REFERENCE (см. 13.2) по крайней мере одно объявление отображения.

Пример — iges2step является именем отображения схемы, которое может содержать объявления для перевода геометрической формы, определенной с использованием набора данных языка EXPRESS, основанного на формате графики IGES, на набор данных, основанный на ИСО 10303-203.

```
SCHEMA_MAP iges2step;
REFERENCE FROM step_schema AS TARGET;
REFERENCE FROM iges_express_schema AS SOURCE;
MAP iges_structure ...
(* другие возможные объявления *)
END_SCHEMA_MAP;
```

Отображение схемы может ссылаться на EXPRESS-схему, другую схему отображения схемы и схему образа схемы, используя синтаксический элемент языка reference_clause (см. 13.2).

Синтаксис:

```
184  schema_map_decl = SCHEMA_MAP schema_map_id ';' reference_clause {
     reference_clause } [ constant_decl ] schema_map_body_element_list
     END_SCHEMA_MAP ';' .
167  reference_clause = REFERENCE FROM schema_ref_or_rename [ '(,
     resource_or_rename { ',' resource_or_rename } ')' ] [ AS ( SOURCE |
     TARGET ) ] ';' .
186  schema_ref_or_rename = [ general_schema_alias_id ':' ]
     general_schema_ref .
```

Правила и ограничения

a) Отображение схемы должно ссылаться по крайней мере на одну EXPRESS-схему, объявленную как источник отображения, используя синтаксический элемент AS SOURCE.

b) Отображение схемы должно ссылаться по крайней мере на одну EXPRESS-схему, объявленную как цель отображения, используя синтаксический элемент AS TARGET.

Пример — Данный пример иллюстрирует обозначение исходной и целевой EXPRESS-схем. EXPRESS-схема `schema_target_one` ***представляет цель отображения. EXPRESS-схема*** `schema_source_one` ***представляет источник отображения; на нее можно ссылаться как на*** `s1` ***в рамках области действия данного отображения схемы*** `map_name`.

```
SCHEMA_MAP map_name;
  REFERENCE FROM schema_target_one AS TARGET;
  REFERENCE FROM s1 : schema_source_one AS SOURCE;
  MAP my_map AS ...
END_SCHEMA_MAP;
```

9.7 Локальное объявление

В разделе отображения могут быть объявлены локальные переменные. Локальные переменные объявляются с помощью элемента языка LOCAL. Локальная переменная видима только в рамках области действия раздела отображения, в котором она объявлена, включая элементы языка IDENTIFIED BY, WHERE и SELECT. При вычислении значения раздела отображения все локальные переменные изначально имеют неопределенное значение, если только их значения не инициализированы в явном виде. Локальным переменным могут присваиваться значения в теле раздела отображения, а в выражениях на них могут даваться ссылки.

Синтаксис:

```
 47  binding_header = [ PARTITION partition_id ';' ] [ from_clause ] {
     local_decl } [ where_clause ] [ identified_by_clause ] [
     ordered_by_clause ] .
129  local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' .
130  local_variable = variable_id { ',' variable_id} ':' parameter_type [
     ':=' expression ] ';' .
```

Правила и ограничения

Синтаксический элемент local_decl не должен использоваться в объявлении образа.

Пример — В данном примере локальной переменной задается значение экземпляра целевого объекта. Атрибут данного объекта модифицируется в теле отображения.

```
SCHEMA_MAP arm2aim;

REFERENCE FROM arm AS SOURCE;
REFERENCE FROM aim AS TARGET;

MAP product_map AS p : product;
  p : product;
 FROM
   ap : arm_product;
 LOCAL
   asc : applied_security_classification;
 END_LOCAL;
 SELECT
   asc := asc@security_classification_map(ap.security);
   asc.items := asc.items + p;
END_MAP;

MAP security_classification_map AS
  asc : applied_security_classification;
  scl : security_classification;
 FROM
   arm_asc : arm_security_classification;
 SELECT
```

```
   asc.items := [];
   asc.classification := scl;
END_MAP;

END_SCHEMA_MAP;
```

9.8 Объявление констант

Константы могут быть определены для использования с помощью элемента языка WHERE в объявлении образа или отображения либо в теле объявления отображения или алгоритма. Константы, имеющие значение объектного типа данных, будут присутствовать в целевом множестве в том случае, если на них ссылаются экземпляры объектного типа данных из данного целевого множества.

Объявления констант определены в подразделе 9.4 ИСО 10303-11.

9.9 Объявление функций

Функции могут быть определены для использования с помощью элемента языка WHERE в объявлении образа или отображения либо в теле объявления отображения. Экземпляр объекта, созданный и возвращенный после вычисления функции, будет присутствовать в целевом множестве в том случае, если на него ссылается атрибут целевого экземпляра из объявления отображения.

Объявления функций определены в пункте 9.5.1 ИСО 10303-11.

9.10 Объявление процедур

Процедуры могут быть определены для использования в телах объявлений функций. Процедуры не должны использоваться непосредственно в теле объявления отображения или образа.

Объявления процедур определены в пункте 9.5.2 ИСО 10303-11.

9.11 Объявление правил

Правила могут быть определены для использования в образе схемы.

Объявления правил определены в пункте 9.6 ИСО 10303-11.

Включение объявлений правил не оказывает влияния ни на модель исполнения, ни на содержимое исходного пространства или пространства образа.

10 Выражения

10.1 Введение

Выражения являются комбинациями операторов, операндов и вызовов функций, с которыми выполняются вычисления для получения значения выражения.

Встроенные функции, определенные в разделе 15, и операторы, определенные в разделе 12 ИСО 10303-11, применимы к настоящему стандарту. Аргументы типов данных образов должны трактоваться как аргументы объектных типов данных. Взаимосвязь между определениями образов и определениями объектов определена в приложении С.

Синтаксис:

```
 81  expression = simple expression [ rel op extended simple expression ] .
169  rel op extended = rel op | IN | LIKE .
168  rel op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:' .
193  simple expression = term { add like op term } .
216  term = factor { multiplication like op factor } .
 84  factor = simple factor [ '**' simple factor ] .
194  simple factor = aggregate initializer | entity constructor |
     enumeration reference | interval | query expression | ( [ unary op ] (
     '(' expression ')' | primary ) ) | case expr | for expr | if expr .
158  primary = literal | ( qualifiable factor { qualifier } ) .
163  qualifiable factor = attribute ref | constant factor | function call |
     general ref | map call | population | target parameter ref |
     view attribute ref | view call .
```

Конструкторы объектов создают экземпляры, являющиеся локальными только в той функции или процедуре, в которой они используются. Экземпляры, созданные конструкторами объектов, не должны образовывать ни целевых, ни исходных множеств.

Процесс вычисления значения выражения управляется приоритетом операторов, составляющих часть выражения. Вычисление значения выражения, заключенного в круглые скобки, производится до того, как оно будет трактоваться как один операнд. Вычисление выражения осуществляется слева направо, при этом сначала выполняются операторы, имеющие наивысший приоритет. В таблице 3 приведены правила приоритетов для всех операторов языка EXPRESS-X. Операторы, расположенные в одной строке таблицы 3, имеют одинаковый приоритет, а строки расположены в порядке убывания приоритета. Операнд, расположенный между двумя операторами с разным приоритетом, относится к оператору с более высоким приоритетом. Операнд, расположенный между двумя операторами с одинаковым приоритетом, относится к левому оператору.

Таблица 3 — Приоритет операторов

Приоритет	Описание	Операторы
1	Ссылки на компоненты	[] . \ :: <- {}
2	Унарные операторы	+ - NOT
3	Возведение в степень	**
4	Умножение/деление	/ * DIV MOD AND |
5	Сложение/вычитание	+ - OR XOR
6	Отношение	= <> <= >= < > :=: :<>: IN LIKE

10.2 Вызов образа

Вызов образа представляет собой выражение, результатом вычисления которого является экземпляр типа данных образа или агрегированная структура экземпляров типа данных образа. Вызов образа обеспечивает средства для доступа к экземпляру (или экземплярам) типа данных образа через имя образа и фактические параметры, соответствующие его связующему экземпляру (если в образе не определен синтаксический элемент IDENTIFIED BY) или значениям выражений, заданных синтаксическим элементом IDENTIFIED BY (если в образе определен синтаксический элемент IDENTIFIED BY). Если в разделе отсутствует синтаксический элемент IDENTIFIED BY, то число, тип и порядок фактических параметров должны соответствовать числу, типу и порядку исходных параметров, указанных в образе в синтаксическом элементе FROM. Если в разделе присутствует синтаксический элемент IDENTIFIED BY, то число, тип и порядок фактических параметров должны соответствовать числу, типу и порядку выражений, заданных в синтаксическом элементе IDENTIFIED BY.

Если ни один связующий экземпляр не соответствует фактическим параметрам вызова образа, то данному вызову присваивается неопределенное значение.

Вызов образа относится к одному разделу образа. Если образ содержит более одного раздела, то может быть задан синтаксический элемент partition_qualification, для того чтобы в явном виде указать раздел, для которого вычисляется значение выражения для данного вызова образа. В противном случае, если синтаксический элемент partition_qualification не задан, то раздел, для которого вычисляется значение выражения для данного вызова образа, должен быть первым разделом в объявлении образа, для которого число и тип аргументов соответствуют параметрам вызова, а синтаксический элемент WHERE возвращает значение TRUE. Для согласования типов данных должны применяться правила совместимости типов данных, определенные в подразделе 12.11 ИСО 10303-11.

Вызов образа, ссылающийся на постоянный раздел, должен иметь пустой список аргументов.

Синтаксис:

```
227  view_call = view_reference [ partition_qualification ] ,(, [
     expression_or_wild { ,,' expression_or_wild } ] ,)' .
153  partition_qualification = ,\' partition_ref .
 82  expression_or_wild = expression | , ' .
```

Пример — Данный пример иллюстрирует использование вызова образа для определения взаимосвязи между двумя типами данных образа. Синтаксический элемент `IDENTIFIED_BY` *в разделе* `person_part` *определяет одно выражение —* `a.creator`*. Поэтому вызовы образа* `approver\person_part` *будут осуществляться с одним аргументов типа* `STRING`*, который является также атрибутом* `creator` *экземпляра объекта* `approval`.

```
SCHEMA_VIEW example;

REFERENCE FROM src_schema;

VIEW approver;
 PARTITION person_part;
  FROM a : approval; p : person;
  WHERE a.creator = p.name;
  IDENTIFIED_BY a.creator;
  SELECT
    approver_id : INTEGER := p.id;
 PARTITION org_part;
  FROM a : approval; o : organization;
  WHERE a.creator = o.name;
  IDENTIFIED_BY a.creator;
  SELECT
    approver_id : INTEGER := o.id;
END_VIEW;

VIEW design order;
  FROM a : approval;
  SELECT
    id : STRING := a.id;
    approved by : approver :=
                  approver\person_part(a.creator);
END_VIEW;
END_SCHEMA_VIEW;

SCHEMA src_schema;

ENTITY approval;
  id : STRING;
  creator : STRING;
END_ENTITY;

ENTITY person;
  name : STRING;
  id : INTEGER;
END_ENTITY;

ENTITY organization;
 id : INTEGER;
 name : STRING;
END_ENTITY;

END_SCHEMA;

(* Исходный набор данных в формате ИСО 10303-21 – см. [2] *)
#1=APPROVAL('a_1','Jones');
#2=APPROVAL('a_2','Smith');
#3=APPROVAL('a_3','Jones');
#4=PERSON('Jones',123);
#5=PERSON('Smith',234);
(* Конечные экземпляры образа в формате ИСО 10303-21 *)
#101=APPROVER(123);
#102=APPROVER(234);
#103=DESIGN_ORDER('a_1',#101);
#104=DESIGN_ORDER('a_2',#102);
#105=DESIGN_ORDER('a_3',#101);
```

10.3 Вызов отображения

Вызов отображения представляет собой выражение, результатом вычисления которого являются экземпляры целевого объекта или агрегированная структура экземпляров объектов. Вызов отображения обеспечивает средства для доступа к значениям, полученным при вычислении отображения (включая зависимое отображение) через имя отображения и фактические параметры, соответствующие его связующему экземпляру (если в данном отображении не определен синтаксический элемент IDENTIFIED BY) или значениям выражений, заданных синтаксическим элементом IDENTIFIED BY (если в отображении определен синтаксический элемент IDENTIFIED BY). Если в отображении отсутствует синтаксический элемент IDENTIFIED BY, то число, тип и порядок фактических параметров должны соответствовать числу, типу и порядку исходных параметров синтаксического элемента FROM в данном разделе. Если в отображении присутствует синтаксический элемент IDENTIFIED BY, то число, тип и порядок фактических параметров должны соответствовать числу, типу и порядку выражений, заданных в синтаксическом элементе IDENTIFIED BY.

Если ни один связующий экземпляр не соответствует фактическим параметрам вызова отображения, то данному вызову присваивается неопределенное значение.

Вызов отображения относится к одному разделу отображения. Если отображение содержит более одного раздела, то может быть задан синтаксический элемент partition qualification, для того чтобы в явном виде указать раздел, для которого вычисляется значение выражения для данного вызова отображения. В противном случае, если синтаксический элемент partition qualification не задан, то раздел, для которого вычисляется значение выражения для данного вызова отображения, должен быть первым разделом в объявлении отображения, для которого число и тип аргументов соответствуют параметрам вызова, а синтаксический элемент WHERE возвращает значение TRUE. Для согласования типов данных должны применяться правила совместимости типов данных, определенные в подразделе12.11 ИСО 10303-11.

Синтаксис:

```
135  map call = [ target parameter ref  '@'] map reference [
     partition qualification ] '(' expression or wild { ','
     expression or wild } ')' .
153  partition qualification = ,\' partition ref .
```

Правила и ограничения

a) Синтаксический элемент target parameter ref должен ссылаться на ссылку на параметр, объявленный в объявлении отображения, на которое ссылается синтаксический элемент map reference.

b) Если в объявлении отображения, на которое ссылается данный вызов отображения, объявлено более одного целевого параметра, то синтаксический элемент target parameter ref @ должен использоваться для идентификации целевого параметра, который должен быть возвращен данным вызовом отображения.

Примеры

1 Данный пример иллюстрирует использование вызова отображения для определения взаимосвязи между объектами в целевой схеме:

```
(* Исходная схема *)
SCHEMA src;
ENTITY approval;
  id : STRING;
  creator : STRING;
END_ENTITY;
END_SCHEMA;

(* Целевая схема *)
SCHEMA tar;
ENTITY person;
  name : STRING;
END_ENTITY;
```

```
ENTITY design_order;
  id : STRING;
  approved_by : person;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP example;
REFERENCE FROM src AS SOURCE;
REFERENCE FROM tar AS TARGET;

MAP person_map AS p : tar.person;
FROM a : approval;
IDENTIFIED_BY a.creator;
SELECT
  p.id := a.creator;
END_MAP;

MAP design_order_map AS d : tar.design_order;
FROM a : approval;
SELECT
  d.id := a.id;
  d.approved_by := p@person_map(a.creator); -- вызов отображения
END_MAP;
END_SCHEMA_MAP;

(* Исходный набор экземпляров, представленный в формате ИСО 10303-21, – см. [2] *)
#1 = APPROVAL('a_1','Miller');
#2 = APPROVAL('a_2','Jones');
#3 = APPROVAL('a_3','Miller');

(* Конечные целевые экземпляры в формате ИСО 10303-21 (см. [2]) *)
#101=PERSON('Miller');
#102=PERSON('Jones');
#103=DESIGN_ORDER('a_1',#101);
#104=DESIGN_ORDER('a_2',#102);
#105=DESIGN_ORDER('a_3',#101);
```

2 Данный пример иллюстрирует использование вызовов отображения, когда разделы не определены в явном виде:

```
SCHEMA source_schema;
TYPE person_select = SELECT (male, female, child);
END_TYPE;

ENTITY male;
 name: STRING;
END_ENTITY;

ENTITY female;
 name : STRING;
END_ENTITY;

ENTITY child;
 name : STRING;
 parents : SET of person_select;
END_ENTITY;
END_SCHEMA;

SCHEMA target_schema;
ENTITY person;
 name : STRING;
END ENTITY;
```

```
ENTITY person_with_parents
 SUBTYPE OF (person);
 parents : SET [1:2] of person;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP source_to_target;
REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP person_map AS
 p : person;
PARTITION p_male;
 FROM m : male;
 SELECT
  p.name := m.name;
PARTITION p_female;
 FROM f : female;
 SELECT
  p.name := f.name;
PARTITION p_child;
 FROM c : child;
 RETURN person_with_parents_map(c);
END_MAP;

MAP person_with_parents_map AS
 p : person_with_parents;
 FROM c: child;
 WHERE SIZEOF (c.parents) > 0; -- объект person_with_parents создается только в том
                               -- случае, когда имеется хотя бы один из родителей.
 SELECT
  p.name := c.name;
  p.parents := FOR EACH par IN c.parents;
                  RETURN person_map(par); -- см. 10.5.
END_MAP;
END_SCHEMA_MAP;

(* Пример значений:
SOURCE:

#1=FEMALE('Julia');
#3=MALE('Richard');
#4=CHILD('Mary', (#1));
#5=CHILD('Paul', (#1,#3));

TARGET:
#6=PERSON('Julia') ;
#7=PERSON_WITH_PARENTS('Mary', (#6));
#8=PERSON('Richard');
#9=PERSON_WITH_PARENTS('Paul', (#8,#6));
*)
```

10.4 Вызовы частичного связывания

Вызов частичного связывания представляет собой вызов образа или отображения, в котором в качестве одного или нескольких аргументов используется элемент языка ' '. Вызов частичного связывания обеспечивает оценку подмножества пространства образа или целевой совокупности элементов в объявлении образа или отображения соответственно.

Элемент языка FROM определяет элементы связующих экземпляров в связующем пространстве вызова образа или отображения (см. 9.2.1). Если в объявлении образа или отображения не задан элемент языка IDENTIFIED BY, то данное объявление определяет функциональное соответствие

между связующими экземплярами и экземплярами из пространства образа или целевой совокупности: в пространстве образа или в целевой совокупности для каждого связующего экземпляра существует уникальный экземпляр. Если в объявлении образа или отображения задан элемент языка IDENTIFIED BY, то данное соответствие не является функциональным и для любого связующего экземпляра не существует уникального экземпляра в пространстве образа или в целевой совокупности. Вместо этого уникальный экземпляр в пространстве образа или в целевой совокупности существует для каждого класса эквивалентности связующих экземпляров, определенного в элементе языка IDENTIFIED BY (см. 9.2.2).

Если элемент языка IDENTIFIED BY не задан, то каждый квалифицированный связующий экземпляр служит ключом для идентификации экземпляра из данного соответствия. Если элемент языка IDENTIFIED BY задан, то любой связующий экземпляр из класса эквивалентности служит ключом для идентификации экземпляра из данного соответствия. Элементы этих двух видов ключей (компоненты ключей) определяются элементом языка FROM и выражениями в IDENTIFIED BY соответственно.

Вызов частичного связывания представляет собой вызов образа или отображения, в котором значения одного или нескольких элементов этих ключей не заданы (в синтаксических структурах это обозначается с помощью элемента языка ' ').

Если элемент языка IDENTIFIED BY не задан, то данный частично заданный ключ соответствует множеству квалифицированных связующих экземпляров, которые соответствуют заданным элементам ключа. Незаданные компоненты ключа (обозначенные с помощью ' ') игнорируются.

Если элемент языка IDENTIFIED BY задан, то данный частично заданный ключ соответствует значениям, полученным в результате вычисления выражения, указанного в элементе языка IDENTIFIED BY. Незаданные компоненты ключа (обозначенные с помощью ' ') игнорируются.

Результатом вызова частичного связывания является множество экземпляров образа (в случае вызова образа) или экземпляров объекта (в случае вызова отображения), соответствующих связующим экземплярам, отобранным по алгоритмам, представленным в данном подразделе.

Пример — В данном примере разные версии, связанные с деталью, собраны вместе с использованием вызова частичного связывания. Возвращенный после вызова частичного связывания объект **version_and_its_product** ***является подмножеством пространства, для которого вторым элементом связующего экземпляра является элемент, равный заданному экземпляру объекта*** **product:**

```
VIEW part;
FROM p : product;
SELECT
  versions : SET OF version_and_its_product
    := version_and_its_product(_, p);
END_VIEW;

VIEW version_and_its_product;
FROM pdf : product_definition_formation; p : product;
WHERE p :=: pdf.of_product;
SELECT
  the version : product_definition_formation := pdf;
END_VIEW;
```

10.5 Выражение FOR

Выражение FOR обеспечивает вычисление агрегированного значения с помощью итераций, управляемых значением индекса или агрегированной структурой. Настоящий стандарт определяет два вида итераций.

- выражение foreach expr проводит итерации, управляемые агрегированной структурой языка EXPRESS, и вычисляет значение объекта expression для каждого элемента данной агрегированной структуры. Все результаты вычислений собираются в агрегированную структуру, которая представляет значение выражения FOR;

- выражение forloop expr проводит итерации, управляемые объектом repeat control (см. подраздел 13.9 ИСО 10303-11), и на каждой итерации вычисляет значение объекта expression.

Если не сделано ни одной итерации, то выражение FOR в качестве результата возвращает пустую агрегированную структуру.

Синтаксис:

```
 89 for expr = FOR ( foreach expr | forloop expr ) .
 85 foreach expr = EACH variable id IN expression [ where clause ] RETURN
    expression .
 86 forloop expr = repeat control RETURN expression .
171 repeat control = [ increment control ] [ while control ] [
    until control ] .
113 increment control = variable id ':=' bound 1 TO bound 2 [ BY increment
    ] .
232 while control = WHILE logical expression .
222 until control = UNTIL logical expression .
```

Правила и ограничения

a) Вычисленное значение объекта `expression` из выражения `foreach expr` должно быть агрегированной структурой языка EXPRESS, пространством или пространством образа.

b) Объявление образа не должно содержать выражение `FOR`.

Элемент языка `foreach expr` неявно объявляет переменную итератора `variable id` (см. синтаксическое правило 85) типа `GENERIC`, видимую в области действия элементов `where clause` и `expression`. Элемент `expression`, расположенный после элемента языка `IN`, должен быть агрегированной структурой языка EXPRESS, над которой выполняются операции. На каждом цикле итерации элемент данной агрегированной структуры связан со своей переменной итератора, начиная с ее значения `LOINDEX` и до значения `HIINDEX` (см. подразделы 15.17 и 15.11 ИСО 10303-11).

Элемент языка `RETURN` из синтаксического правила 85 определяет выражение, вычисляемое для каждого элемента на данной итерации. В свою очередь данное выражение вычисляется в среде, связывающей переменную итератора с каждым значением исходной агрегированной структуры. Результат каждого вычисления добавляется к агрегированной структуре целевого атрибута так, как если бы был применен оператор объединения (см. пункт 12.6.3 ИСО 10303-11) с агрегированной структурой целевого атрибута в качестве его левого операнда и результатом вычисления выражения в качестве правого операнда.

Необязательный элемент `where clause` из синтаксического правила 85 определяет выражение, которое должно возвращать значение типа `LOGICAL` или неопределенное значение. Выражение, следующее за элементом языка `RETURN`, только вычисляется, а его результат включается в агрегированную структуру результата только в том случае, если рассчитанным значением элемента `where clause` является `TRUE`.

Примеры

1 В данном примере целевой объект **component** ***является отображением исходного объекта*** **product_definition**, ***а все экземпляры объекта*** **product_definition_name**, ***ссылающиеся на один экземпляр объекта*** **product_definition**, ***сгруппированы в целевом атрибуте*** **component.names**:

```
(* Исходная схема *)
SCHEMA source_schema;
ENTITY product_definition;
  product_name : STRING;
  description : STRING;
END_ENTITY;
ENTITY product_definition_name;
  name    : STRING;
  of_product_definition : product_definition;
END_ENTITY;
END_SCHEMA;
(* Целевая схема *)
SCHEMA target_schema;
ENTITY component;
  names : SET [0:?] OF STRING;
  product_name  : STRING;
  description  : STRING;
END_ENTITY;
END_SCHEMA;
```

```
(* Определение отображения *)
SCHEMA_MAP example;
REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;
MAP component_map AS c : component;
FROM pd : product_definition;
SELECT
  c.description := pd.description;
  c.product_name := pd.product_name;
  c.names := FOR EACH pdn_instance
               IN EXTENT('SOURCE_SCHEMA.PRODUCT_DEFINITION_NAME');
               WHERE pdn_instance.of_product_definition :=: pd;
          RETURN pdn_instance.name;
END_MAP;
END_SCHEMA_MAP;
```

2 Данный пример иллюстрирует использование вложенных выражений FOR. *При этом пример 1 изменяется следующим образом:*

```
(* Исходная схема *)
SCHEMA source_schema;

ENTITY product_definition;
  product_name : STRING;
  description : STRING;
END_ENTITY;
ENTITY product_definition_name;
  name : STRING;
  of_product_definition : product definition;
END_ENTITY;
ENTITY product_definition_value;
  of_pdn : product_definition_name;
  val : STRING;
END_ENTITY;
END_SCHEMA;

(* Целевая схема *)
SCHEMA target_schema;
ENTITY component;
  values : SET [0:?] OF SET [0:?] OF STRING;
  product_name  : STRING;
  description  : STRING;
END_ENTITY;
END_SCHEMA;
```

Все экземпляры объекта product_definition_value, *ссылающиеся на один экземпляр объекта* product_definition_name, *сгруппированы вместе и представлены во внутренней агрегированной структуре атрибута* component.values. *Это выглядит следующим образом:*

```
(* Определение отображения *)
SCHEMA_MAP example;

REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP component_map AS c : component;
FROM pd : product_definition;
SELECT
  c.description := pd.description;
  c.product name := pd.product_name;
  c.values := FOR EACH pdn_instance
                IN EXTENT('SOURCE_SCHEMA.PRODUCT_DEFINITION_NAME') ;
                WHERE pdn_instance.of_product_definition :=: pd;
           RETURN FOR EACH pdv_instance
                        IN EXTENT('SOURCE_SCHEMA.PRODUCT_DEFINITION_VALUE');
                    WHERE pdv_instance.of_pdn :=: pdn_instance;
                   RETURN pdv_instance.val;
END_MAP;
END_SCHEMA_MAP;
```

10.6 Выражение IF

Выражение IF обеспечивает условную оценку элементов языка expression на основе оценки элемента logical expression из синтаксического правила 110. Если ни один из элементов logical expression не имеет значения TRUE и не задан элемент языка ELSE, то атрибуту отображения или образа присваивается неопределенное значение.

Синтаксис:

```
110 if expr = IF logical expression THEN expression { ELSIF
    Logical expression expression } [ ELSE expression ] END IF .
```

10.7 Выражение CASE

Выражение CASE обеспечивает условную оценку элементов языка expression в соответствии с шаблоном выполнения оператора CASE языка EXPRESS (см. подраздел 13.4 ИСО 10303-11). Если ни один элемент языка OTHERWISE не задан и ни один из элементов case expr action не имеет значения TRUE, то атрибуту отображения или образа присваивается неопределенное значение.

Синтаксис:

```
56  case expr = CASE selector OF { case expr action } [ OTHERWISE ,:'
    expression ] END CASE .
57  case expr action = case label { ,' case label } ,:' expression ,;' .
```

Правила и ограничения

Выражение CASE не должно использоваться в образах схем, относящихся к классу соответствия 1.

***Пример — Данный пример иллюстрирует использование выражения* CASE:**

```
SCHEMA source_schema;
ENTITY approval;
 status : STRING;
END_ENTITY;
END_SCHEMA;

SCHEMA target_schema;
ENTITY my_approval;
 status : INTEGER;
END_ENTITY;
END_SCHEMA;

SCHEMA_MAP mapping_example;
REFERENCE FROM source_schema AS SOURCE;
REFERENCE FROM target_schema AS TARGET;

MAP approval_map AS ma : my_approval;
FROM a : approval;
SELECT
  ma.status := CASE a.status OF
                 'approved'     : 1;
                 'not yet approved' : -1;
                 'disapproved'  : 0;
                 OTHERWISE      : 2;
               END_CASE;
END_MAP;
END_SCHEMA_MAP;
```

10.8 Оператор прямого пути

Оператор прямого пути (::) создает агрегированную структуру экземпляров объектов, на которые ссылается значение элемента attribute ref из синтаксического правила 88. Если необязательный элемент extent reference из синтаксического правила 154 задан, то созданная агрегированная

структура должна содержать экземпляры только тех объектов, тип данных которых соответствует элементу `extent reference` или одному из его подтипов.

Синтаксис:

```
 88  forward path qualifier = '::' attribute ref [ path condition ] .
154  path condition = '{' extent reference [ '|' logical expression ] ,}' .
```

Правила и ограничения

a) Оператор прямого пути не должен использоваться в образах схем, относящихся к классу соответствия 1.

b) Переменная, имя которой совпадает с именем элемента `extent reference`, является неявно объявленной в области действия оператора прямого пути.

Примечания

1 Данная переменная не должна быть объявлена где-либо еще, и она не будет существовать за пределами данного оператора.

2 Использование оператора прямого пути иллюстрирует пример в 10.9.

Если логическое выражение `logical expression` из синтаксического правила 154 задано, то элементы выбираются по очереди из пространства, на которое дана ссылка, и связываются с неявно объявленной переменной. Затем вычисляется значение логического выражения `logical expression` с учетом данного связывания. Для каждого такого связывания переменной проверяется, является ли значением логического выражения `logical expression` TRUE, и если это так, то данный элемент добавляется к результату, в противном случае данный элемент к результату не добавляется.

Примечание — Функция **unnest**, использованная в приведенном ниже примере, принимает один аргумент произвольного типа данных (включая вложенные агрегированные структуры) и возвращает агрегированную структуру, элементы которой не относятся к агрегированному типу данных. Например обращение unnest(**[[a],[b,c],[[d]]]**) возвращает результат **[a,b,c,d]**. Определение функции **unnest** приведено в приложении E.

Пример — Для некоторого множества элементов a, ссылки на объект* product *и атрибута экземпляров* of_product *выражение* result:=a::of_product{product} *эквивалентно следующей EXPRESS-спецификации:

```
LOCAL
  result : AGGREGATE OF GENERIC := [];
  tmp : AGGREGATE OF GENERIC := [];
END_LOCAL;
tmp := unnest(a);
REPEAT i := 1 TO HIINDEX(tmp);
  result := result + QUERY(e <* unnest(tmp[i].of_product)|
                    'SCHEMA_NAME.PRODUCT' IN TYPEOF(e));
END_REPEAT;
result := unnest (result);
```

Выражение result:=a::x эквивалентно следующей EXPRESS-спецификации:

```
result := [];
tmp := unnest(a);
REPEAT i := 1 TO HIINDEX(tmp);
  result := result + unnest(tmp[i].x);
END_REPEAT;
result := unnest(result)
```

10.9 Оператор обратного пути

Оператор обратного пути (<-) создает агрегированную структуру экземпляров объектов, используя выражение, расположенное справа от данного оператора. Вычисление выражения `a <-x ( b | c )` в результате дает агрегированную структуру экземпляров объектов, в которой для каждого ее элемента `e` справедливы следующие утверждения:

- атрибут `x` элемента `e` ссылается на элемент невложенного `a`;

- элемент e имеет тип данных b;
- результатом логического выражения c является TRUE в среде, в которой неявная переменная b связана с e.

Синтаксис:

```
 43  backward path qualifier = '<-' [ attribute ref ] path condition .
154  path condition = '{' extent reference [ '|' logical expression ] ,}' .
```

Правила и ограничения

a) Оператор обратного пути не должен использоваться в образах схем, относящихся к классу соответствия 1.

b) Элемент attribute ref должен быть определен в каком-либо из частичных объектных типов данных в каждом экземпляре из пространства аргумента.

c) Переменная, имя которой совпадает с именем элемента extent reference, является неявно объявленной в области действия оператора обратного пути.

Если идентификатор a представляет совокупность элементов, то выражение result := a<-x{b} эквивалентно следующей EXPRESS-спецификации:

```
LOCAL
  result : AGGREGATE OF GENERIC := [];
  tmp : AGGREGATE OF GENERIC := [];
END LOCAL;
result := [];
tmp := unnest(a);
REPEAT i := 1 TO HIINDEX(tmp);
  result := result + QUERY(e <* USEDIN(tmp[i], '') |
            ('SCHEMA_NAME.B' IN TYPEOF(e))
            AND (tmp[i] IN e.x));
END REPEAT;
```

Выражение a<-{b} эквивалентно следующей EXPRESS-спецификации:

```
result := [];
tmp := unnest(a);
REPEAT i := 1 TO HIINDEX(tmp);
   result := result + QUERY(e <* USEDIN(tmp[i], '') |
                       ('SCHEMA_NAME.B' IN TYPEOF(e)));
END REPEAT;
```

Примечание — Функция unnest, использованная в приведенных выше EXPRESS-спецификациях, принимает один аргумент произвольного типа данных (включая вложенные агрегированные структуры) и возвращает агрегированную структуру, элементы которой не относятся к агрегированному типу данных. Например, обращение unnest([[a],[b,c],[[d]]]) возвращает результат [a,b,c,d]. Определение функции **unnest** приведено в приложении E.

Пример — В данном примере операторы пути используются для вычисления исходной агрегированной структуры в цикле создания экземпляра объекта. Исходная агрегированная структура содержит все экземпляры объекта **document_file**, ***ссылающегося на экземпляр объекта*** **representation_type** ***со значением атрибута*** **name** **'digital'**, ***и на который имеется ссылка как на атрибут*** **documentation_ids** ***экземпляра объекта*** **product_definition_with_associated_documents**, ***который в свою очередь ссылается на экземпляр исходного объекта*** **product_ definition_formation:**

```
SCHEMA document_schema;

ENTITY folder;
   name : STRING;
END_ENTITY;

ENTITY file;
    name     : STRING;
    location : folder;
END_ENTITY;
END_SCHEMA;
```

```
SCHEMA source_schema;

ENTITY product_definition_formation;
 id : STRING;
 name : STRING;
END_ENTITY;

ENTITY product_definition_with_associated_documents;
 documentation_ids : SET OF document_file;
 formation : product_definition_formation;
END_ENTITY;

ENTITY document_file;
 name : STRING;
 representation_type : document_representation;
END_ENTITY;

ENTITY document_representation;
 name : STRING;
END_ENTITY;

END SCHEMA;

SCHEMA_MAP example2;
REFERENCE FROM document_schema AS TARGET;
REFERENCE FROM source_schema AS SOURCE;
MAP document_map AS
  folder : folder;
  files: AGGREGATE OF file;
  FROM pdf : product_definition_formation;
    FOR EACH f IN
      pdf<-formation{product_definition_with_associated_documents}
            ::documentation_ids{document_file
            | document_file.representation_type.name = 'digital'}
      INDEXING i;
      SELECT
        files[i].name := f.name;
        files[i].location := folder;
        folder.name := pdf.name;
END_MAP;
END_SCHEMA_MAP;
```

11 Встроенная функция

11.1 Универсальная функция EXTENT

```
FUNCTION EXTENT ( R : STRING ) : SET OF GENERIC;
```

Функция EXTENT формирует пространство, содержащее множество всех экземпляров из данной совокупности, которые имеют тип данных, заданный параметром функции.

Параметр R является строкой, содержащей имя объектного типа данных или типа данных образа. Эти имена квалифицируются с помощью имени схемы, содержащей определение указанного типа данных.

Результатом является множество, содержащее все экземпляры, имеющие заданный параметром функции объектный тип данных или тип данных образа. Ошибочным является задание параметром типа данных, который не является типом данных образа или объектным типом данных, определенным в исходной схеме.

Пример — Пространство объектного типа данных **action** ***в схеме*** **automotive_design** ***формируется с помощью следующего вызова функции*** **EXTENT:**

```
EXTENT('AUTOMOTIVE_DESIGN.ACTION');
```

12 Области действия и видимости

12.1 Введение

Объявление на языке EXPRESS-X создает идентификатор, который может быть использован для ссылки на объявленный элемент в других частях образа схемы или отображения схемы, а также в

других образах или отображениях схем. Некоторые конструкции языка EXPRESS-X неявно объявляют элементы, присваивая им идентификаторы. Можно сказать, что элемент является видимым в тех областях, где на идентификатор, присвоенный объявленному элементу, может быть дана ссылка. Ссылка на элемент должна даваться только там, где его идентификатор является видимым.

Некоторые элементы языка EXPRESS-X определяют некоторую область (блок текста), называемую областью действия элемента. Область действия ограничивает видимость объявленных в ней идентификаторов. Области действия могут быть вложенными, т. е. элемент языка EXPRESS-X, установивший некоторую область действия, может быть включен в область действия другого элемента. Существуют ограничения того, какие элементы могут присутствовать в области действия конкретного элемента языка EXPRESS-X. Такие ограничения задаются в терминах синтаксиса элементов языка.

В таблице 4 представлены элементы языка EXPRESS-X, для каждого из которых в соответствующем подразделе определены ограничения на область действия (если таковые имеются) и область видимости объявленного идентификатора как в общих терминах, так и в деталях. Для используемых в языке EXPRESS-X элементов, определенных в языке EXPRESS (см. ИСО 10303-11), ограничения заданной области действия и области видимости объявленного идентификатора соответствуют ограничениям, установленным в языке EXPRESS (см. подраздел 10.3 ИСО 10303-11).

В языке EXPRESS-X применяются общие правила для областей действия и видимости, установленные в языке EXPRESS (см. подразделы 10.1, 10.2, пункт 10.2.1 ИСО 10303-11).

Таблица 4 — Области действия и идентификаторы элементов языка EXPRESS-X

Элемент	Область действия	Идентификатор
Цикл создания экземпляра объекта	+	+*
Выражение FOR	+	+*
Зависимое отображение, Отображение	+	+
Зависимый образ, Образ	+	+
Исходный параметр	—	+
Целевой параметр	—	+
Оператор пути	+	+
*Идентификатор является неявно объявленной переменной в рамках области действия данного объявления.		

12.2 Образ схемы

Область видимости: Объявление образа схемы является видимым во всех других образах схемы.

Область действия: Объявление образа схемы определяет новую область действия. Данная область действия начинается с ключевого слова `SCHEMA VIEW` и заканчивается ключевым словом `END SCHEMA VIEW`, которое завершает объявление образа схемы.

Объявления: Объявлять идентификаторы в рамках области действия объявления образа схемы могут следующие элементы:

- константы;
- функции;
- процедуры;
- правила;
- образы.

12.3 Отображение схемы

Область видимости: Идентификатор отображения схемы является видимым во всех других образах схемы и отображениях схемы.

Примечание — На объявления отображений в отображении схемы не должен ссылаться образ схемы.

Область действия: Объявление отображения схемы определяет новую область действия. Данная область действия начинается с ключевого слова `SCHEMA MAP` и заканчивается ключевым словом `END SCHEMA MAP`, которое завершает объявление отображения схемы.

Объявления: Объявлять идентификаторы в рамках области действия отображения схемы могут следующие элементы:

- константы;
- функции;
- процедуры;
- образы;
- зависимые образы;
- отображения;
- зависимые отображения;
- правила.

12.4 Образ и зависимый образ

Область видимости: Идентификатор образа или зависимого образа является видимым в области действия образа или отображения схемы, в которой он объявлен. Идентификатор образа или зависимого образа остается видимым во всех внутренних областях действия, в которых данный идентификатор не объявлен повторно.

Область действия: Объявление образа или зависимого образа определяет новую область действия. Данная область действия начинается с ключевого слова `VIEW (DEPENDENT VIEW)` и заканчивается ключевым словом `END VIEW (END DEPENDENT VIEW)`, которое завершает данное объявление образа.

Объявления: Объявлять идентификаторы в рамках области действия объявления образа могут следующие элементы:

- метки раздела;
- элементы языка `FROM`;
- элементы языка `IDENTIFIED BY`;
- атрибуты образа.

12.5 Метка раздела образа

Область видимости: Правила области видимости для метки раздела образа идентичны правилам области видимости для образа (см. 12.4).

12.6 Идентификатор атрибута образа

Область видимости: Идентификатор атрибута образа является видимым в области действия образа, в котором он объявлен.

12.7 Выражение FOR

Область видимости: Неявно объявленная переменная итератора является видимой в выражении, которое следует после ключевого слова `RETURN` в выражении `FOR`, и в элементе языка `WHERE` из синтаксического правила 85.

Область действия: Выражение `FOR` определяет новую область действия. Данная область действия является видимой для выражения, которое завершается ключевым словом `RETURN` в выражении `FOR`.

Объявления: Объявлять идентификаторы в рамках области действия выражения `FOR` могут следующие элементы:

- выражения `FOR`;
- выражения `QUERY`.

12.8 Отображение и зависимое отображение

Область видимости: Идентификатор отображения или зависимого отображения является видимым в области действия образа или отображения схемы, в которой он объявлен. Идентификатор отображения или зависимого отображения остается видимым во всех внутренних областях действия, в которых данный идентификатор не объявлен повторно.

Область действия: Объявление отображения или зависимого отображения определяет новую область действия. Данная область действия начинается с ключевого слова `MAP (DEPENDENT MAP)`

и заканчивается ключевым словом END MAP (END DEPENDENT MAP), которое завершает данное объявление отображения.

Объявления: Объявлять идентификаторы в рамках области действия объявления отображения или зависимого отображения могут следующие элементы:

- выражения FOR;
- выражения QUERY;
- циклы создания экземпляра объекта;
- элементы языка FROM;
- элементы языка IDENTIFIED BY;
- атрибуты.

12.9 Элемент языка FROM

Область видимости: Идентификатор исходного параметра является видимым в разделе, в котором он объявлен и в образах или отображениях подтипов данного образа либо отображения.

12.10 Цикл создания экземпляра объекта

Область видимости: Неявно объявленные идентификаторы циклов создания экземпляров объектов являются видимыми в рамках представителей элемента языка SELECT, за которым следует цикл создания экземпляра объекта. Область действия заканчивается ключевым словом END MAP или предыдущим циклом создания экземпляра объекта.

Объявления: Выражение QUERY может объявлять идентификаторы в рамках области действия цикла создания экземпляра объекта.

12.11 Оператор пути

Область видимости: В рамках пути, заданного выражением path condition, атрибуты квалификатора типа данных и его супертипов являются видимыми так же, как ключевое слово SELF. Кроме того, все идентификаторы, объявленные вне данного выражения, являются видимыми.

Область действия: Оператор пути определяет новую вложенную область действия, заданную выражением языка path condition, которая является областью действия объектного типа данных, к которому относится квалификатор типа данных (см. пункт 10.3.5 ИСО 10303-11). Новая область действия начинается с синтаксического элемента '|' и заканчивается закрывающей фигурной скобкой '}'.

Объявления: Выражение QUERY может объявлять идентификаторы в рамках области действия пути, заданного выражением path condition.

13 Спецификация интерфейса

13.1 Введение

Спецификации интерфейсов дают возможность элементам, объявленным в одной внешней схеме, образе схемы или отображении схемы, быть видимыми в другом образе или отображении схемы.

13.2 Элемент языка REFERENCE

Элемент языка REFERENCE дает возможность элементам, объявленным в одной схеме, образе схемы или отображении схемы, быть видимыми в другом образе или отображении схемы.

Спецификация интерфейса REFERENCE позволяет сделать видимыми в данном образе схемы следующие элементы, объявленные во внешней схеме, образе схемы или отображении схемы:

- образы;
- зависимые образы;
- отображения;
- зависимые отображения;
- константы;
- объекты;
- типы данных;
- функции;

- процедуры;
- правила.

Спецификация интерфейса REFERENCE определяет имя внешней схемы, образа схемы или отображения схемы и, необязательно, имена объявленных в них элементов языка EXPRESS или EXPRESS-X. Если имена элементов не заданы, то все элементы, объявленные во внешней схеме, образе схемы или отображении схемы, являются видимыми в данном образе или отображении схемы.

Синтаксис:

```
167  reference_clause = REFERENCE FROM schema_ref_or_rename [ '('
     resource_or_rename { ',' resource_or_rename } ')' ] [ AS ( SOURCE |
     TARGET ) ] ';' .
186  schema_ref_or_rename = [ general_schema_alias_id ':' ]
     general_schema_ref .
103  general_schema_ref = schema_ref | schema_map_ref | schema_view_ref .
174  resource_or_rename = resource_ref [ AS rename_id ] .
```

Правила и ограничения

a) Элемент rename_id должен быть уникальным в области действия ссылающейся на него схемы, включая все другие ссылочные идентификаторы. Ссылающаяся схема должна ссылаться на объявление с помощью своего элемента rename_id.

b) Элемент general_schema_ref должен быть уникальным в области действия ссылающейся на него схемы.

c) Образ схемы не должен ссылаться на объявление отображения.

Пример — Данный пример иллюстрирует обозначение исходной EXPRESS-схемы и ссылку на ее ресурсы из другой схемы. На ресурс **your_view_decl** ***дается ссылка из схемы*** **other_map_schema**, ***и он переименовывается в*** **my_view_decl** ***для использования в образе данной схемы:***

```
SCHEMA_VIEW my_view_schema;
  REFERENCE FROM automotive_design;
  REFERENCE FROM other_map_schema (your_view_decl AS my_view_decl);
END_SCHEMA_VIEW;
```

Приложение А
(обязательное)

Регистрация информационного объекта

Для обеспечения однозначного обозначения информационного объекта в открытой системе настоящему стандарту присвоен следующий идентификатор объекта:

{ iso standard 10303 part(14) version(1) }

Смысл данного обозначения установлен в ИСО/МЭК 8824-1 и описан в ИСО 10303-1.

Приложение В
(обязательное)

Синтаксис языка EXPRESS-X

В настоящем приложении определены лексические элементы языка и грамматические правила, которым данные элементы должны подчиняться.

Примечание — Прямое применение данного определения синтаксиса приведет к неоднозначности при построении синтаксических анализаторов. Данное определение разработано для представления информации, относящейся к использованию идентификаторов. Интерпретированные идентификаторы определяют лексические элементы, являющиеся ссылками на объявленные идентификаторы, и поэтому не должны трактоваться как простые идентификаторы (simple_id). Разработчик синтаксического анализатора должен обеспечить разрешение ссылок на идентификаторы и получение необходимого ссылочного лексического элемента для проверки грамматических правил.

Все грамматические правила языка EXPRESS, определенные в ИСО 10303-11, приложение А, являются также грамматическими правилами языка EXPRESS-X. Помимо грамматических правил языка EXPRESS к грамматическим правилам языка EXPRESS-X относятся грамматические правила, определенные в данном приложении.

В.1 Лексические элементы

Приведенные ниже правила определяют лексические элементы, используемые в языке EXPRESS-X. За исключением тех случаев, когда это явно установлено в синтаксических правилах, никакие пробелы или комментарии не должны присутствовать в тексте, относящемся к отдельному синтаксическому правилу, представленному в настоящем приложении.

В.1.1 Ключевые слова

В настоящем подразделе установлены правила, используемые для представления ключевых слов языка EXPRESS-X.

Примечание — В настоящем подразделе используется соглашение, по которому каждое ключевое слово представлено синтаксическим правилом, содержащим в левой части данное ключевое слово, записанное с использованием символов верхнего регистра (прописных букв).

Зарезервированными словами языка EXPRESS-X являются зарезервированные слова языка EXPRESS, а также ключевые слова и имена встроенных функций языка EXPRESS-X. Зарезервированные слова языка EXPRESS-X не должны использоваться как идентификаторы.

```
 1 DEPENDENT_MAP = 'dependent_map' .
 2 EACH = 'each' .
 3 ELSEIF = 'elseif' .
 4 END_DEPENDENT_MAP = 'end_dependent_map' .
 5 END_MAP = 'end_map' .
 6 END_SCHEMA_MAP = 'end_schema_map' .
 7 END_SCHEMA_VIEW = 'end_schema_view' .
 8 END_VIEW = 'end_view' .
 9 EXTENT = 'extent' .
10 IDENTIFIED_BY = 'identified_by' .
11 INDEXING = 'indexing' .
12 MAP = 'map' .
13 ORDERED_BY = 'ordered_by' .
14 PARTITION = 'partition' .
15 SCHEMA_MAP = 'schema_map' .
16 SCHEMA_VIEW = 'schema_view' .
17 SOURCE = 'source' .
18 TARGET = 'target' .
19 VIEW = 'view' .
```

В.1.2 Классы символов

```
20 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
21 letter = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j'
          | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't'
          | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' .
22 simple_id = letter { letter | digit | '_'}.
```

B.1.3 Интерпретированные идентификаторы

Примечание — Все интерпретированные идентификаторы языка EXPRESS являются также интерпретированными идентификаторами языка EXPRESS-X. Кроме того, в языке EXPRESS-X используются следующие интерпретированные идентификаторы.

```
23 partition_ref = partition_id .
24 map_ref = map_id .
25 schema_map_ref = schema_map_id .
26 schema_view_ref = schema_view_id .
27 source_parameter_ref = source_parameter_id .
28 source_schema_ref = schema_ref .
29 target_parameter_ref = target_parameter_id .
30 target_schema_ref = schema_ref .
31 view_attribute_ref = view_attribute_id .
32 view_ref = view_id .
```

B.2 Грамматические правила

Представленные в настоящем разделе правила определяют, как рассмотренные выше лексические элементы и лексические элементы языка EXPRESS могут объединяться в конструкции языка EXPRESS-X. Пробелы и комментарии языка EXPRESS могут быть помещены между любыми двумя лексемами в данных правилах. Первичным синтаксическим правилом языка EXPRESS-X является `syntax_x`.

```
33 abstract_supertype_declaration = ABSTRACT SUPERTYPE [
   subtype_constraint ] .
34 actual_parameter_list = '(' parameter { ',' parameter } ')' .
35 add_like_op = '+' | '-' | OR | XOR .
36 aggregate_initializer = '[' [ element { ',' element } ] ']' .
37 aggregate_source = simple_expression .
38 aggregate_type = AGGREGATE [ ':' type_label ] OF parameter_type .
39 aggregation_types = array_type | bag_type | list_type | set_type .
40 algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
41 array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ] base_type .
42 assignment_stmt = general_ref { qualifier } ':=' expression ';' .
43 backward_path_qualifier = '<-' [ attribute_ref ] path_condition .
44 bag_type = BAG [ bound_spec ] OF base_type .
45 base_type = aggregation_types | simple_types | named_types .
46 binary_type = BINARY [ width_spec ] .
47 binding_header = [ PARTITION partition_id ';' ] [ from_clause ]
   [ local_decl ] [ where_clause ] [ identified_by_clause ]
   [ ordered_by_clause ] .
48 boolean_type = BOOLEAN .
49 bound_1 = numeric_expression .
50 bound_2 = numeric_expression .
51 bound_spec = '[' bound_1 ':' bound_2 ']' .
52 built_in_constant = CONST_E | PI | SELF | '?' .
53 built_in_function = ABS | ACOS | ASIN | ATAN | BLENGTH | COS
   | EXISTS | EXTENT | EXP | FORMAT | HIBOUND | HIINDEX | LENGTH
   | LOBOUND | LOINDEX | LOG | LOG2 | LOG10 | NVL | ODD | ROLESOF
   | SIN | SIZEOF | SQRT | TAN | TYPEOF | USEDIN | VALUE | VALUE_IN
   | VALUE_UNIQUE .
54 built_in_procedure = INSERT | REMOVE .
55 case_action = case_label { ',' case_label } ':' stmt .
56 case_expr = CASE selector OF { case_expr_action } [ OTHERWISE ':'
   expression ] END_CASE .
57 case_expr_action = case_label { ',' case_label } ':' expression ';' .
58 case_label = expression .
59 case_stmt = CASE selector OF { case_action } [ OTHERWISE ':' stmt ]
   END_CASE ';' .
60 compound_stmt = BEGIN stmt { stmt } END ';' .
61 constant_body = constant_id ':' base_type ':=' expression ';' .
62 constant_decl = CONSTANT constant_body { constant_body } END_CONSTANT
   ';' .
63 constant_factor = built_in_constant | constant_ref .
```

```
 64 constant_id = simple_id .
 65 declaration = function_decl | procedure_decl .
 66 dependent_map_decl = DEPENDENT_MAP map_id AS target_parameter {
    target_parameter } [ map_subtype_of_clause ] dep_map_partition {
    dep_map_partition } END_DEPENDENT_MAP ';' .
 67 dependent_view_decl = VIEW view_id ':' base_type ';' binding_header
    RETURN expression { binding_header RETURN expression } END_VIEW ';' .
 68 dep_binding_decl = dep_from_clause [ where_clause ] [
    ordered_by_clause ] .
 69 dep_from_clause = FROM dep_source_parameter ';' {
    dep_source_parameter ';' } .
 70 dep_map_decl_body = dep_binding_decl map_project_clause .
 71 dep_map_partition = [ PARTITION partition_id ':'
    ] dep_map_decl_body .
 72 dep_source_parameter = source_parameter_id { ','
    source_parameter_id} ':' ( simple_types | type_reference ) .
 73 domain_rule = [ label ':' ] logical_expression .
 74 element = expression [ ':' repetition ] .
 75 entity_constructor = entity_reference '(' [ expression
    { ',' expression } ] ')' .
 76 entity_id = simple_id .
 77 entity_instantiation_loop = FOR instantiation_loop_control ';'
    map_project_clause .
 78 entity_reference = [ ( source_schema_ref | target_schema_ref |
    schema_ref ) '.' ] entity_ref .
 79 enumeration_reference = [ type_reference '.' ] enumeration_ref .
 80 escape_stmt = ESCAPE ';' .
 81 expression = simple_expression [ rel_op_extended simple_expression ] .
 82 expression_or_wild = expression | '*' .
 83 extent_reference = source_entity_reference | view_reference .
 84 factor = simple_factor [ '**' simple_factor ] .
 85 foreach_expr = EACH variable_id IN expression [ where_clause ]
    RETURN expression .
 86 forloop_expr = repeat_control RETURN expression .
 87 formal_parameter = parameter_id { ',' parameter_id } ':'
    parameter_type .
 88 forward_path_qualifier = '::' attribute_ref [ path_condition ] .
 89 for_expr = FOR ( foreach_expr   forloop_expr ) .
 90 from_clause = FROM source_parameter ';' { source_parameter ';' } .
 91 function_call = ( built_in_function   function_ref ) [
    actual_parameter_list ] .
 92 function_decl = function_head [ algorithm_head ] stmt { stmt }
    END_FUNCTION ';' .
 93 function_head = FUNCTION function_id [ '(' formal_parameter { ';'
    formal_parameter } ')' ] ':' parameter_type ';' .
 94 function_id = simple_id .
 95 generalized_types = aggregate_type | general_aggregation_types |
    generic_type .
 96 general_aggregation_types = general_array_type | general_bag_type |
    general_list_type   general_set_type .
 97 general_array_type = ARRAY [ bound_spec ] OF [ OPTIONAL ] [ UNIQUE ]
    parameter_type .
 98 general_attribute_qualifier = '.' ( attribute_ref |
    view_attribute_ref ) .
 99 general_bag_type = BAG [ bound_spec ] OF parameter_type .
100 general_list_type = LIST [ bound_spec ] OF [ UNIQUE ]
    parameter_type .
101 general_ref = parameter_ref | variable_ref | source_parameter_ref .
102 general_schema_alias_id = schema_id | schema_map_id | schema_view_id .
103 general_schema_ref = schema_ref | schema_map_ref | schema_view_ref .
104 general_set_type = SET [ bound_spec ] OF parameter_type .
105 generic_type = GENERIC [ ':' type_label ] .
```

```
106 group_qualifier = '\' entity_ref .
107 identified_by_clause = IDENTIFIED_BY id_parameter ';'
    { id_parameter ';' } .
108 id_parameter = [ id_parameter_id ',:' ] expression .
109 id_parameter_id = simple_id .
110 if_expr = IF logical_expression THEN expression { ELSIF
    logical_expression expression } [ ELSE expression ] END_IF .
111 if_stmt = IF logical_expression THEN stmt { stmt } [ ELSE stmt {
    stmt } ] END_IF ',;' .
112 increment = numeric_expression .
113 increment_control = variable_id ',:=' bound_1 TO bound_2
    [ BY increment ] .
114 index = numeric_expression .
115 index_1 = index .
116 index_2 = index .
117 index_qualifier = ',[, index_1 ' ,:' index_2] ,]' .
118 instantiation_foreach_control = EACH variable_id IN expression {
    AND variable_id IN expression } [ INDEXING variable_id ] .
119 instantiation_loop_control = instantiation_foreach_control |
    repeat_control .
120 integer_type = INTEGER .
121 interval = '{' interval_low interval_op interval_item interval_op
    interval_high '}' .
122 interval_high = simple_expression .
123 interval_item = simple_expression .
124 interval_low = simple_expression .
125 interval_op = '<' | '<=' .
126 label = simple_id .
127 list_type = LIST [ bound_spec ] OF { UNIQUE } base_type .
128 literal = binary_literal | integer_literal | logical_literal |
    real_literal | string_literal .
129 local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' .
130 local_variable = variable_id { ',' variable_id } ':'
    parameter_type [ ':=' expression ] ';' .
131 logical_expression = expression .
132 logical_literal = FALSE | TRUE | UNKNOWN .
133 logical_type = LOGICAL .
134 map_attribute_declaration = [ target_parameter_ref [
    index_qualifier ] { group_qualifier } '.' ] attribute_ref {
    index_qualifier } ':=' expression ';' .
135 map_call = [ target_parameter_ref '@'] map_reference |
    partition_qualification ] '(' expression_or_wild { ','
    expression_or_wild } ')' .
136 map_decl = MAP map_id AS target_parameter ';' { target_parameter
    ';' } ( map_subtype_of_clause subtype_binding_header map_decl_body
    ) | ( binding_header map_decl_body { binding_header map_decl_body
    } ) END_MAP ';' .
137 map_decl_body = ( entity_instantiation_loop {
    entity_instantiation_loop } )
    | map_project_clause
    | ( RETURN expression ';' ) .
138 map_id = simple_id .
139 map_project_clause = SELECT { map_attribute_declaration } .
140 map_reference = [ schema_map_ref '.' ] map_ref .
141 map_subtype_of_clause = SUBTYPE OF '(' map_reference ')' ';' .
142 multiplication_like_op = '*' | '/' | DIV | MOD | AND | '||' .
143 named_types = entity_reference | type_reference | view_reference .
144 null_stmt = ';' .
145 number_type = NUMBER .
146 numeric_expression = simple_expression .
147 one_of = ONEOF '(' supertype_expression
    { ',' supertype_expression } ')' .
```

```
148 ordered_by_clause = ORDERED_BY expression { ',' expression } ';' .
149 parameter = expression .
150 parameter_id = simple_id .
151 parameter_type = generalized_types | named_types | simple_types .
152 partition_id = simple_id .
153 partition_qualification = '\' partition_ref .
154 path_condition = '{' extent_reference [ '|' logical_expression ]
    '}' .
155 path_qualifier = forward_path_qualifier | backward_path_qualifier .
156 population = entity_reference .
157 precision_spec = numeric_expression .
158 primary = literal | ( qualifiable_factor { qualifier } ) .
159 procedure_call_stmt = ( built_in_procedure | procedure_ref ) [
    actual_parameter_list ] ';' .
160 procedure_decl = procedure_head [ algorithm_head ] { stmt }
    END_PROCEDURE ';' .
161 procedure_head = PROCEDURE procedure_id [ '(' [ VAR ]
    formal_parameter { ';' [ VAR ] formal_parameter } ')' ] ';' .
162 procedure_id = simple_id .
163 qualifiable_factor = attribute_ref | constant_factor |
    function_call | general_ref | map_call | population |
    target_parameter_ref | view_attribute_ref | view_call .
164 qualifier = general_attribute_qualifier | group_qualifier |
    index_qualifier | path_qualifier .
165 query_expression = QUERY '(' variable_id '<*' aggregate_source '|'
    logical_expression ')' .
166 real_type = REAL [ '(' precision_spec ')' ] .
167 reference_clause = REFERENCE FROM schema_ref_or_rename [ '(',
    resource_or_rename { ',' resource_or_rename } ')' ] [ AS ( SOURCE |
    TARGET ) ] ';' .
168 rel_op = '<' | '>' | '<=' | '>=' | '<>' | ':=' | ':<>:' | ':=:' .
169 rel_op_extended = rel_op | IN | LIKE .
170 rename_id = constant_id | entity_id | function_id | procedure_id |
    type_id .
171 repeat_control = [ increment_control ] [ while_control ] [
    until_control ] .
172 repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT
    ';' .
173 repetition = numeric_expression .
174 resource_or_rename = resource_ref [ AS rename_id ] .
175 resource_ref = constant_ref | entity_ref | function_ref |
    procedure_ref | type_ref | view_ref | map_ref .
176 return_stmt = RETURN [ '(' expression ')' ] ';' .
177 root_view_decl = VIEW view_id [ supertype_constraint ] ';'
    binding_header SELECT view_attr_decl_stmt_list { binding_header
    SELECT view_attr_decl_stmt_list } END_VIEW ';' .
178 rule_decl = rule_head [ algorithm_head ] { stmt } where_clause
    END_RULE ';' .
179 rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')'
    ';' .
180 rule_id = simple_id .
181 schema_id = simple_id .
182 schema_map_body_element = function_decl | procedure_decl |
    view_decl | map_decl | dependent_map_decl | rule_decl .
183 schema_map_body_element_list = schema_map_body_element {
    schema_map_body_element } .
184 schema_map_decl = SCHEMA_MAP schema_map_id ';' reference_clause {
    reference_clause } [ constant_decl ] schema_map_body_element_list
    END_SCHEMA_MAP ';' .
185 schema_map_id = simple_id .
186 schema_ref_or_rename = [ general_schema_alias_id ':' ]
    general_schema_ref .
```

```
187 schema_view_body_element = function_decl | procedure_decl |
    view_decl | rule_decl .
188 schema_view_body_element_list = schema_view_body_element {
    schema_view_body_element } .
189 schema_view_decl = SCHEMA_VIEW schema_view_id ';' {
    reference_clause } [ constant_decl ] schema_view_body_element_list
    END_SCHEMA_VIEW ';' .
190 schema_view_id = simple_id .
191 selector = expression .
192 set_type = SET [ bound_spec ] OF base_type .
193 simple_expression = term { add_like_op term } .
194 simple_factor = aggregate_initializer | entity_constructor |
    enumeration_reference | interval | query_expression | ( [ unary_op ]
    ( '(' expression ')' | primary ) ) | case_expr | for_expr | if_expr .
195 simple_types = binary_type | boolean_type | integer_type |
    logical_type | number_type | real_type | string_type .
196 skip_stmt = SKIP ';' .
197 source_entity_reference = entity_reference .
198 source_parameter = source_parameter_id ':' extent_reference .
199 source_parameter_id = simple_id .
200 stmt = assignment_stmt | case_stmt | compound_stmt | escape_stmt |
    if_stmt | null_stmt | procedure_call_stmt | repeat_stmt |
    return_stmt | skip_stmt .
201 string_literal = simple_string_literal | encoded_string_literal .
202 string_type = STRING [ width_spec ] .
203 subtype_binding_header = [ PARTITION partition_id ';' ]
    where_clause .
204 subtype_constraint = OF '(' supertype_expression ')' .
205 subtype_declaration = SUBTYPE OF '(' view_ref { ',' view_ref } ')' .
206 subtype_view_decl = VIEW view_id subtype_declaration ';'
    subtype_binding_header SELECT view_attr_decl_stmt_list {
    subtype_binding_header SELECT view_attr_decl_stmt_list }
    END_VIEW ';' .
207 supertype_constraint = abstract_supertype_declaration |
    supertype_rule .
208 supertype_expression = supertype_factor { ANDOR supertype_factor } .
209 supertype_factor = supertype_term { AND supertype_term } .
210 supertype_rule = SUPERTYPE [ subtype_constraint ] .
211 supertype_term = view_ref | one_of | '(' supertype_expression ')' .
212 syntax_x = schema_map_decl | schema_view_decl .
213 target_entity_reference = entity_reference { '&' entity_reference } .
214 target_parameter = target_parameter_id { ',' target_parameter_id }
    ':' [ AGGREGATE [ bound_spec ] OF ] target_entity_reference .
215 target_parameter_id = simple_id ';' .
216 term = factor { multiplication_like_op factor } .
217 type_id = simple_id .
218 type_label = type_label_id | type_label_ref .
219 type_label_id = simple_id .
220 type_reference = [ schema_ref '.' ] type_ref .
221 unary_op = '+' | '-' | NOT .
222 until_control = UNTIL logical_expression .
223 variable_id = simple_id .
224 view_attribute_decl = view_attribute_id ':' [ OPTIONAL ] [
    source_schema_ref '.' ] base_type ':=' expression ';' .
225 view_attribute_id = simple_id .
226 view_attr_decl_stmt_list = { view_attribute_decl } .
227 view_call = view_reference [ partition_qualification ] '(' [
    expression_or_wild { ',' expression_or_wild } ] ')' .
228 view_decl = ( root_view_decl | dependent_view_decl |
    subtype_view_decl ) .
229 view_id = simple_id .
230 view_reference = [ ( schema_map_ref | schema_view_ref ) '.' ]
```

```
    view_ref .
231 where_clause = WHERE domain_rule ';' { domain_rule ';' }
232 while_control = WHILE logical_expression .
233 width = numeric_expression .
234 width_spec = '(' width ')' [ FIXED ] .
```

B.3 Список перекрестных ссылок

DEPENDENT_MAP | 66
EXTENT | 53
IDENTIFIED_BY | 137
MAP | 136
ORDERED_BY | 148
PARTITION | 47 71 203
SCHEMA_MAP | 184
SCHEMA_VIEW | 189
SELECT | 139 177 206
VIEW | 67 177 206
abstract_supertype_declaration | 207
actual_parameter_list | 91 159
add_like_op | 193
aggregate_initializer | 194
aggregate_source | 165
aggregate_type | 95
aggregation_types | 45
algorithm_head | 92 160 178
array_type | 39
assignment_stmt | 200
backward_path_qualifier | 155
bag_type | 39
base_type | 41 44 61 127 192 224
binary_type | 195
binding_header | 67 136 177
boolean_type | 195
bound_1 | 51 113
bound_2 | 51 113
bound_spec | 41 44 97 99 100 104 127 192 214
built_in_constant | 63
built_in_function | 91
built_in_procedure | 159
case_action | 59
case_expr | 194
case_expr_action | 56
case_label | 55 57
case_stmt | 200
compound_stmt | 200
constant_body | 62
constant_decl | 40 184 189
constant_factor | 163
constant_id | 61 170
declaration | 40
dependent_map_decl | 182
dependent_view_decl | 228
dep_binding_decl | 70
dep_from_clause | 68
dep_map_decl_body | 71
dep_map_partition | 66
dep_source_parameter | 69
domain_rule | 231
element | 36
entity_constructor | 194
entity_id | 170
entity_instantiation_loop | 137

entity_reference | 75 143 156 197 213
enumeration_reference | 194
escape_stmt | 200
expression | 42 56 57 58 61 67 74 75 82 85 86 108 110 118 130 132 134 137 148 149 176 191 194 224
expression_or_wild | 135 227
extent_reference | 154 198
factor | 216
foreach_expr | 89
forloop_expr | 89
formal_parameter | 93 161
forward_path_qualifier | 155
for_expr | 194
from_clause | 47
function_call | 163
function_decl | 65 182 187
function_head | 92
function_id | 93 170
generalized_types | 151
general_aggregation_types | 95
general_array_type | 96
general_attribute_qualifier | 164
general_bag_type | 96
general_list_type | 96
general_ref | 42 163
general_schema_alias_id | 186
general_schema_ref | 186
general_set_type | 96
generic_type | 95
group_qualifier | 134 164
identified_by_clause | 47
id_parameter | 107
id_parameter_id | 108
if_expr | 194
if_stmt | 200
increment | 113
increment_control | 171
index | 115 116
index_1 | 117
index_2 | 117
index_qualifier | 134 164
instantiation_foreach_control | 119
instantiation_loop_control | 77
integer_type | 195
interval | 194
interval_high | 121
interval_item | 121
interval_low | 121
interval_op | 121
label | 73
list_type | 39
literal | 158
local_decl | 40 47
local_variable | 129
logical_expression | 73 110 121 154 165 222 232
logical_literal | 128
logical_type | 195
map_attribute_declaration | 139
map_call | 163
map_decl | 182
map_decl_body | 136

map_id | 66 136
map_project_clause | 70 77 137
map_reference | 135 141
map_subtype_of_clause | 66 136
multiplication_like_op | 216
named_types | 45 151
null_stmt | 200
number_type | 195
numeric_expression | 49 50 112 114 157 173 233
one_of | 211
ordered_by_clause | 47 68
parameter | 34
parameter_id | 87
parameter_type | 38 87 93 97 99 100 104 130
partition_id | 47 71 203
partition_qualification | 135 227
path_condition | 43 88
path_qualifier | 164
population | 163
precision_spec | 166
primary | 194
procedure_call_stmt | 200
procedure_decl | 65 182 187
procedure_head | 160
procedure_id | 161 170
qualifiable_factor | 158
qualifier | 42 158
query_expression | 194
real_type | 195
reference_clause | 184 189
rel_op | 169
rel_op_extended | 81
rename_id | 174
repeat_control | 86 119 172
repeat_stmt | 200
repetition | 74
resource_or_rename | 167
resource_ref | 174
return_stmt | 200
root_view_decl | 228
rule_decl | 182 187
rule_head | 178
rule_id | 179
schema_id | 102
schema_map_body_element | 183
schema_map_body_element_list | 184
schema_map_decl | 212
schema_map_id | 102 184
schema_ref_or_rename | 167
schema_view_body_element | 188
schema_view_body_element_list | 189
schema_view_decl | 212
schema_view_id | 102 189
selector | 56 59
set_type | 39
simple_expression | 37 81 122 123 124 146
simple_factor | 84
simple_types | 45 72 151
skip_stmt | 200
source_entity_reference | 83
source_parameter | 90
source_parameter_id | 72 198

stmt | 55 59 60 92 111 160 172 178
string_literal | 128
string_type | 195
subtype_binding_header | 136 206
subtype_constraint | 33 210
subtype_declaration | 206
subtype_view_decl | 228
supertype_constraint | 177
supertype_expression | 147 204 211
supertype_factor | 208
supertype_rule | 207
supertype_term | 209
target_entity_reference | 214
target_parameter | 66 136
target_parameter_id | 214
term | 193
type_id | 170
type_label | 38 105
type_label_id | 218
type_reference | 72 79 143
unary_op | 194
until_control | 171
variable_id | 85 113 128 130 165
view_attribute_decl | 226
view_attribute_id | 224
view_attr_decl_stmt_list | 177 206
view_call | 163
view_decl | 182 187
view_id | 67 177 206
view_reference | 83 143 227
where_clause | 47 68 85 178 203
while_control | 171
width | 234
width_spec | 46 202

Приложение С
(обязательное)

Алгоритм преобразования текста с языка EXPRESS-X на язык EXPRESS

В настоящем приложении представлено преобразование совокупности объявлений образов в совокупность объявлений объектов на языке EXPRESS, обеспечивающую представление результатов выполнения операторов языка EXPRESS-X. Данное преобразование представлено в форме алгоритма, принимающего текст объявления образа в качестве входного параметра и производящего в результате своей работы текст объявления объекта. Данный алгоритм представлен здесь только с целью спецификации языка EXPRESS-X и не предусматривает какой-либо конкретной реализации.

Предполагается, что преобразованные объекты существуют в схеме с уникальными именами, с которой все необходимые внешние объявления связаны с помощью интерфейсов.

Описание алгоритма

a) Если объявление образа представляет зависимый образ (то есть не определяет никаких атрибутов образа), то пропустить данное объявление.

b) Заменить ключевое слово VIEW на ENTITY.

c) Удалить элементы языка FROM, WHERE, IDENTIFIED_BY и ORDERED_BY. Элементы WHERE удалить только в заголовке, но не удалять в ограничениях.

d) Удалить ключевое слово SELECT.

e) Если объявление образа содержит разделы, то полностью удалить все разделы, кроме объявления первого раздела; удалить ключевое слово PARTITION и идентификатор раздела (если он имеется) из объявления первого раздела.

f) Удалить оператор присваивания и следующее за ним выражение у каждого атрибута образа.

g) Заменить ключевое слово END_VIEW на END_ENTITY.

Примеры

1 Следующее объявление образа:

```
VIEW a ABSTRACT SUPERTYPE;
PARTITION one;
FROM b:one; c:two;
WHERE cond1;
      cond2;
SELECT
  x : attr1 : = expression1;
  y : attr2 : = expression2;
PARTITION two;
FROM d:two; e:three;
WHERE cond3;
      cond4;
SELECT
  x : attr1 : = expression3;
  y : attr2 : = expression4;
END_VIEW;
```

преобразуется в следующее объявление объекта на языке EXPRESS:

```
ENTITY a ABSTRACT SUPERTYPE;
  x : attr1;
  y : attr2;
END_ENTITY;
```

2 Следующее объявление образа:

```
VIEW b SUBTYPE OF (a);
PARTITION one;
WHERE cond5;
SELECT
  z : attr3 := expression5;
PARTITION two;
WHERE cond6;
SELECT
```

```
  z : attr3 := expression6;
WHERE
  WR2 : rule_expression2;
END_VIEW;
```

преобразуется в следующее объявление объекта на языке EXPRESS:

```
ENTITY b SUBTYPE OF (a);
  z : attr3;
WHERE
  WR2 : rule_expression2;
END_ENTITY;
```

Приложение D
(справочное)

Вопросы реализации

D.1 Полное отображение

Реализация выполняет полное отображение в том случае, если она удовлетворяет следующим критериям:

- программа отображения принимает на входе один или несколько наборов исходных данных и производит один или несколько выходных наборов данных (отображений) или пространств образов (образов);
- выходные наборы данных производятся из входных наборов данных с помощью выполнения и вычисления всех объявлений образов (VIEW) отображений и отображений (MAP);
- каждый экземпляр из входных наборов данных отображается на выходные наборы данных (отображения) или пространства образов (образы) так, как указано в схеме отображения.

D.2 Отображение по запросам

Реализация выполняет отображение по запросам в том случае, если она удовлетворяет следующим критериям:

- программа отображения принимает на входе один или несколько наборов исходных данных;
- только указанные в запросах экземпляры целевых данных производятся из наборов входных данных с помощью выполнения и вычисления соответствующих объявлений образов или отображений.

Примечание — В настоящем стандарте не определено, как объявления образов и отображений выбираются для отображения по запросам.

D.3 Поддержка проверки ограничений

Реализация поддерживает проверку ограничений в том случае, если в ней реализованы положения, описанные в подразделе 9.6 ИСО 10303-11, по отношению к экземплярам объектов в целевых наборах данных и экземплярам образов в пространствах образов.

Примечание — Проверка ограничений не влияет на выполнение отображения.

D.4 Поддержка обновлений

Выполнение обновлений невозможно в тех случаях, когда имеет место любое из следующих положений:

- образ/целевой объект производится/отображается из двух или более исходных объектов с помощью одной операции.

Пример — Образ/целевой объект **person_in_dept** ***соответствует исходным объектам*** **person** ***и*** **department**, ***если значением объединяющего их условия*** **person.id = department.person_id** ***является*** **TRUE**;

- дубликаты, т. е. экземпляры, имеющие одинаковые значения атрибутов, существующие в исходных данных, удаляются из образа/целевых данных;
- атрибуты образов/целевые атрибуты получены/отображены из элементов исходной схемы с помощью математических выражений, которые не являются математически обратимыми;
- схема образа/целевая схема определяет дополнительные подтипы данных, которые не существуют в исходной схеме (схемах);
- подтипы данных, определенные в исходной схеме (схемах), переносятся в схему образа/целевую схему (т. е. не содержатся в ней);
- порядок сортировки исходных атрибутов типа данных AGGREGATE аннулируется в схеме образа/целевой схеме;
- дубликаты (с точки зрения эквивалентности значений) элементов исходных атрибутов типа данных AGGREGATE исключаются из схемы образа/целевой схемы;
- один исходный объект соответствует сети взаимосвязанных образов/целевых объектов (через взаимосвязи или эквивалентность значений атрибутов*).

* Данный вид взаимосвязи сопоставим со взаимосвязями типа «первичный ключ — внешний ключ» в реляционной модели данных.

Приложение Е
(справочное)

Функция unnest оператора пути

Следующий текст на языке EXPRESS реализует функцию unnest, на которую даны ссылки в 10.8 и 10.9:

```
FUNCTION unnest(src : GENERIC) : AGGREGATE OF GENERIC;
LOCAL
  result : AGGREGATE OF GENERIC := [];
  tmp : AGGREGATE OF GENERIC;
END_LOCAL;
  IF SIZEOF(['LIST', 'BAG', 'SET', 'ARRAY', 'AGGREGATE'] * TYPEOF(src)) > 0
  THEN
    REPEAT i :=1TO HIINDEX(src);
      tmp := unnest(src[i]);
      REPEAT j := 1 TO HIINDEX(tmp);
        result := result + tmp[j];
      END_REPEAT;
    END_REPEAT;
  ELSE
    IF SIZEOF(['STRING', 'BINARY', 'LOGICAL', 'NUMBER'] * TYPEOF(src)) = 0
    THEN -- entity instance
      result := result + src;
    END_IF;
  END_IF;
  RETURN (result);
END_FUNCTION;
```

Приложение F
(справочное)

Семантика таблицы отображений

Язык EXPRESS-X может быть использован в качестве альтернативной спецификации ссылочных путей в таблицах отображений прикладного протокола STEP. В данном приложении показано, как это может быть сделано с помощью примеров прототипов на языке EXPRESS-X для всех конструкций ссылочных путей из таблиц отображений, описанных в Руководстве по разработке спецификаций отображений, подраздел 9.1 «Использование символов». В данных примерах использована спецификация с большей функциональностью за счет применения конструкции VIEW языка EXPRESS-X. При этом отображение осуществляется от прикладной интерпретированной модели (ПИМ) к прикладной эталонной модели (ПЭМ). Отображение в обратном направлении от ПЭМ к ПИМ может быть сделано более наглядно с помощью процедурной конструкции MAP языка EXPRESS-X. Направление ссылочных путей предполагается таким же, как в таблицах отображений прикладных протоколов, — от ПЭМ (сверху и слева) к ПИМ (вниз и направо)

F.1 Разделительные символы

Символ супертипа "->" показывает, что элемент со стороны ПЭМ, расположенный слева от этого символа, должен быть супертипом для элемента со стороны ПИМ, расположенного справа от символа. При отображении на ПЭМ данный символ показывает направление от подтипа к супертипу, которое не требует отдельной спецификации на языке EXPRESS-X, так как такое наследование определено в языке EXPRESS.

Символ подтипа "<-" показывает, что элемент со стороны ПЭМ, расположенный слева от этого символа, должен быть подтипом для элемента со стороны ПИМ, расположенного справа от символа. При отображении на ПЭМ данный символ показывает требование к подтипу, который должен быть задан на языке EXPRESS-X с помощью элемента языка WHERE, содержащего ссылку TYPEOF, следующим образом:

Таблица отображений:

```
cc_design_approval <-
approval_assignment
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... aa:approval_assignment; ...
WHERE ... 'CONFIG_CONTROL_DESIGN.CC_DESIGN_APPROVAL' IN TYPEOF(aa); ...
```

Следует отметить,что приведенный выше фрагмент во всех случаях эквивалентен следующему:

```
VIEW ...;
FROM ... aa:cc_design_approval; ...
```

Поэтому в случае наличия обоих ограничителей связанных супертипов или связанных подтипов только наиболее конкретизированный подтип должен появиться в качестве исходного типа данных, как показано в следующем примере:

Таблица отображений:

```
product_definition_usage ->
assembly_component_usage ->
next_assembly_usage_occurrence
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... nauo:next_assembly_usage_occurrence; ...
```

Если существует несколько таких специфичных подтипов в цепочке связанных ограничителей супертипов и подтипов, то может быть использовано условие TYPEOF или условие равенства экземпляров:

Таблица отображений:

```
representation_item ->
measure_representation_item
{ measure_representation_item <-
  measure_with_unit ->
  length_measure_with_unit }
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... mri:measure_representation_item; ...
WHERE ... 'FEATURE_BASED_PROCESS_PLANNING.LENGTH_MEASURE_WITH_UNIT' IN
  TYPEOF(mri);
```

или

```
VIEW ...;
FROM ... lmwu:length_measure_with_unit; ...
WHERE 'FEATURE_BASED_PROCESS_PLANNING.MEASURE_REPRESENTATION_ITEM' IN
  TYPEOF(lmwu);
```

или

```
VIEW ...;
FROM ... mri:measure_representation_item;
  lmwu:length_measure_with_unit; ...
WHERE ... mri :=: lmwu; ...
```

Символ ссылки на атрибут "->" показывает, что элемент со стороны ПЭМ, расположенный слева от этого символа, является ссылкой на атрибут, тип данных которого определяет элемент со стороны ПИМ, расположенный справа от символа. При отображении на ПЭМ данный символ показывает направление от типа данных к атрибуту данного типа. Символ ссылки от атрибута "<-" является обратным, показывая в отображении на ПЭМ направление от атрибута к типу данных этого атрибута. В общем случае оба эти символа могут быть определены на языке EXPRESS-X с помощью условия равенства экземпляров:

Таблица отображений:

```
approval_assignment.assigned_approval ->
approval
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... aa:approval_assignment; a:approval; ...
WHERE ... aa.assigned_approval :=: a; ...
```

Аналогично ограничителям супертипов и подтипов, цепочки ограничителей ссылок на атрибуты могут быть объединены так, что только «самые верхние» ссылки должны появиться в качестве исходных типов данных. Кроме того, может быть желательно включить также начальный и/или конечный типы данных из полного ссылочного пути, для того чтобы зафиксировать их:

Таблица отображений:

```
cc_design_person_and_organization_assignment <-
person_and_organization_assignment
person_and_organization_assignment.assigned_person_and_organization ->
person_and_organization
person_and_organization.the_person ->
person
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... poa:cc_design_person_and_organization_assignment; p:person; ...
WHERE ... poa.assigned_person_and_organization.the_person :=: p; ...
```

F.2 Символы агрегированных структур

Символ агрегированной структуры "[i]" определен в языке EXPRESS-X с помощью ограничения "IN", а символ "[n]" определен аналогично тому, как в языке EXPRESS-X определена ссылка на атрибут:

Таблица отображений:

```
property_definition_representation
property_definition_representation.used_representation ->
representation
representation.items[i] ->
representation_item ->
measure_representation_item
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... pdr:property_definition_representation;
         mri:measure_representation_item;
WHERE ... mri IN pdr.used_representation.items;
```

Таблица отображений:

```
representation
representation.items[2] ->
representation_item ->
geometric_representation_item ->
placement ->
axis2_placement_3d
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... r:representation; ap:axis2_placement_3d;
WHERE ... r.items[2] :=: ap;
```

F.3 Знак равенства

Символ знака равенства "=" используется для задания типа у типа данных SELECT или для ограничения значения атрибута, имеющего простой тип данных. В первом случае функции знака равенства очень похожи на функции символов задания супертипов и подтипов данных, и он может комбинироваться с ними в цепочках конкретизации или обобщения, в которых, как было отмечено выше, только наиболее конкретизированные типы данных из цепочки должны быть определены как исходные типы данных для элемента языка VIEW:

Таблица отображений:

```
approval_date_time
approval_date_time.date_time ->
date_time_select
date_time_select = date
date ->
calendar_date
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... adt:approval_date_time; cd: calendar_date; ...
WHERE ... adt.date_time :=: cd;
```

В случае, когда символ "=" используется для ограничения значения атрибута, имеющего простой тип данных (т. е. заключенного в квадратные скобки), ограничение в языке EXPRESS-X задается непосредственно с помощью элемента языка WHERE:

Таблица отображений:

```
representation
representation.items[i] ->
( representation_item
  representation_item.name = 'diameter' )
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... r:representation; ri:representation_item; ...
WHERE ... ri IN r.items; ri.name = 'diameter'; ...
```

F.4 Круглые скобки

Символы круглых скобок "(" и ")" используются для заключения в них отдельных опций в ссылочном пути. При отображении на ПЭМ это означает, что любые опции, присутствующие в ПИМ, в результате будут отражены на ПЭМ. В языке EXPRESS-X это задается с помощью элемента языка PARTITION:

Таблица отображений:

```
person_and_organization
( person_and_organization.the_person ->
  person <-
  personal_address.people[i]
  personal_address <-
  address )
( person_and_organization.the_organization ->
  organization <-
  organizational_address <-
  address )
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
PARTITION p;
FROM ... po:person_and_organization; pa:personal_address; ...
WHERE ... po.the_person IN pa.people;
...
PARTITION o;
FROM ... po:person_and_organization; oa:organizational_address; ...
WHERE ... po.the_organization IN oa.organizations;
...
```

F.5 Квадратные скобки

Символы квадратных скобок "[" и "]" используются для заключения в них ветвей ссылочного пути, все из которых должны существовать. В языке EXPRESS-X это задается с помощью добавления к образу спецификаций всех таких подветвей:

Таблица отображений:

```
[ mapped_item
  mapped_item.mapping_target ->
  representation_item ]
[ mapped_item
  mapped_item.mapping_source ->
  representation_map
  representation_map.mapping_origin ->
  representation_item ]
```

Фрагмент на языке EXPRESS-X:

```
VIEW ...;
FROM ... mi:mapped_item; source:representation_item;
         target:representation_item; ...
WHERE ... mi.mapping_target :=: target;
          mi.mapping_source.mapping_origin :=: source; ...
```

F.6 Пример

Таблица отображений объекта `circular_closed_profile` на `numeric_parameter`, представляющий атрибут `diameter`:

```
circular_closed_profile <-
shape_aspect
shape_definition = shape_aspect
shape_definition
characterized_definition = shape_definition
characterized_definition <-
property_definition.definition
property_definition <-
property_definition_representation.definition
{ property_definition_representation ->
  shape_definition_representation }
property_definition_representation
property_definition_representation.used_representation ->
{ representation ->
  shape_representation ->
  shape_representation_with_parameters }
representation
representation.items[i] ->
{ representation_item
  representation_item.name = 'diameter' }
representation_item ->
measure_representation_item
{ measure_representation_item <-
  measure_with_unit ->
  length_measure_with_unit }
```

Фрагмент на языке EXPRESS-X:

```
VIEW find_diameter;
FROM ccp:circular_closed_profile;
     sdr:shape_definition_representation;
     srwp:shape_representation_with_parameters;
     mri:measure_representation_item;
     lmwu:length_measure_with_unit;
IDENTIFIED_BY ccp;
WHERE sdr.definition.definition :=: ccp;
      sdr.used_representation :=: srwp;
      mri IN srwp.items;
      mri.name = 'diameter';
      lmwu :=: mri;
RETURN
  lmwu.value_component;
END_VIEW;
```

Приложение ДА
(справочное)

Сведения о соответствии ссылочных международных стандартов национальным стандартам Российской Федерации

Таблица ДА.1

Обозначение ссылочного международного стандарта	Степень соответствия	Обозначение и наименование соответствующего национального стандарта
ИСО/МЭК 8824-1:2002	IDT	ГОСТ Р ИСО/МЭК 8824-1—2001 «Информационная технология. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Спецификация основной нотации»
ИСО 10303-1:1994	IDT	ГОСТ Р ИСО 10303-1—99 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 1. Общие представления и основополагающие принципы»
ИСО 10303-11:2004	IDT	ГОСТ Р ИСО 10303-11—2009 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 11. Методы описания. Справочное руководство по языку EXPRESS»
ИСО/МЭК 10646-1:2000	—	*

* Соответствующий национальный стандарт отсутствует. До его утверждения рекомендуется использовать перевод на русский язык данного международного стандарта. Перевод данного международного стандарта находится в Федеральном информационном фонде технических регламентов и стандартов.

Примечание — В настоящей таблице использовано следующее условное обозначение степени соответствия стандартов:

- IDT — идентичные стандарты.

Библиография

[1] Wirth, Niklaus, What can we do about the unnecessary diversity of notations for syntactic definitions? Communications of the ACM, November 1977, vol. 20, no. 11, p. 822

[2] ISO 10303-21:1994, Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure

УДК 656.072:681.3:006.354 ОКС 25.040.40

Ключевые слова: автоматизация производства, системы автоматизации, интеграция систем автоматизации, данные об изделиях, представление данных, обмен данными, методы описания, искусственные языки, языки моделирования, язык EXPRESS-X

Редактор *Л.С. Зимилова*
Корректор *П.М. Смирнов*
Компьютерная верстка *Л.А. Круговой*

Подписано в печать 08.02.2016. Формат 60 ×84 1/8.
Усл. печ. л. 8,37 Тираж 30 экз. Зак. 3894.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
gost.e-ecolog.ru gost.e-ecolog.ru

gost.e-ecolog.ru