ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Системы автоматизации производства и их интеграция ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 12

Методы описания
Справочное руководство по языку EXPRESS-I

Издание официальное

БЗ 10—2000/304

ГОССТАНДАРТ РОССИИ
Москва

gost.ecology.spb.ru ecolog.ru

Предисловие

1 РАЗРАБОТАН Всероссийским научно-исследовательским институтом стандартизации (ВНИИстандарт) при участии Научно-технического центра «ИНТЕГРО-Д»

ВНЕСЕН Техническим комитетом по стандартизации ТК 431 «CALS-технологии»

2 ПРИНЯТ И ВВЕДЕН В ДЕЙСТВИЕ Постановлением Госстандарта России от 14 ноября 2000 г. № 292-ст

3 Настоящий стандарт содержит полный аутентичный текст международного стандарта ИСО/ТО 10303-12-97 «Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 12. Методы описания. Справочное руководство по языку EXPRESS-I»

4 ВВЕДЕН ВПЕРВЫЕ

© ИПК Издательство стандартов, 2001

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Госстандарта России

Содержание

1 Область применения 1
2 Нормативные ссылки 1
3 Определения 2
3.1 Термины, определенные в ГОСТ Р ИСО 10303-1 2
3.2 Термины, определенные в ГОСТ Р ИСО 10303-11 2
3.3 Термины, определенные в ИСО 10303-31 2
3.4 Другие определения 2
4 Требования соответствия 3
4.1 Формальные спецификации, записанные на EXPRESS-I 3
4.2 Реализации EXPRESS-I 3
5 Основные принципы 3
6 Элементы языка 4
6.1 Набор символов 4
6.2 Зарезервированные слова 6
6.3 Знаки 7
6.4 Идентификаторы и ссылки 8
7 Именованные области значений 9
7.1 Область значений объекта 9
7.2 Область значений перечисления 9
7.3 Область значений выбора 9
7.4 Область значений типа 10
8 Значения и экземпляры 10
8.1 Базовые значения 10
8.2 Агрегатные значения 13
8.3 Простой экземпляр 13
8.4 Экземпляр типа 14
8.5 Экземпляр выбора 14
8.6 Экземпляр перечисления 14
8.7 Экземпляр объекта 15
8.8 Экземпляр константы 18
8.9 Экземпляр данных схемы 18
8.10 Отображение модели 19
9 Определение абстрактного контрольного примера 19
9.1 Контекст 19
9.2 Параметры 20
9.3 Контрольный пример 21
9.4 Цель теста 22
9.5 Реализация теста 23
10 Интерфейсы 24
10.1 Интерфейс экземпляра схемы 24
10.2 Ссылка на схему 24
10.3 Ссылки на данные контекста 25
11 Область действия и видимость 25
11.1 Правила области действия 26
11.2 Правила видимости 26
11.3 Правила для явного элемента 27
12 Отображение из EXPRESS в EXPRESS-I 32
12.1 Отображение EXPRESS-схемы 32
12.2 Отображение простых типов данных из EXPRESS 33
12.3 Отображение агрегатных типов данных 34
12.4 Отображение определенного типа данных из EXPRESS 35
12.5 Отображение перечисляемого типа из EXPRESS 35
12.6 Отображение выбираемого типа из EXPRESS 35
12.7 Отображение EXPRESS-константы 36
12.8 Отображение EXPRESS-объекта 36
12.9 Отображение атрибутов EXPRESS-объекта 37

12.10 Отображение супертипов и подтипов 41
Приложение А Описание синтаксиса EXPRESS-I 44
А.1 Лексемы 44
А.2 Лексические элементы 47
А.3 Интерпретируемые идентификаторы 48
А.4 Грамматические правила 48
А.5 Список перекрестных ссылок 54
Приложение В Заявка о соответствии реализации протоколу (ЗСРП) 63
В.1 Синтаксический анализатор языка EXPRESS-I 63
Приложение С Регистрация информационного объекта 64
Приложение D Синтаксис спецификации языка 64
D.1 Синтаксис спецификации 64
D.2 Нотация специального символа 65
Приложение Е Некоторые контрольные примеры 66
E.1 Контрольный пример 1 66
E.2 Контрольный пример 2 67
E.3 Контрольный пример 3 68
E.4 Контрольный пример 4 69
Приложение F Замечания по применению стандарта 71
F.1 Примеры EXPRESS-данных 71
F.2 Абстрактные контрольные примеры 71
F.3 Объектные базы 71
F.4 Примеры данных, отличных от EXPRESS 72
Приложение G Технические подходы 73
G.1 Абстрактные контрольные примеры 73
G.2 Связь с EXPRESS 73
G.3 Ссылки на предметы 73
G.4 Агрегации 73
G.5 Строковые значения 73
G.6 Тестирование и принятие модели 74
G.7 Расширение возможностей контрольного примера 74
G.8 Соответствие языку EXPRESS 74
G.9 Опытная апробация 74
G.10 Расширения алфавита 74
G.11 Отображение супертипов 74
G.12 Комментарии по голосованию за CD−1995 75
Приложение Н Библиография 76
Предметный указатель 77

Введение

Стандарты серии ГОСТ Р ИСО 10303 распространяются на машинно-ориентированное представление данных об изделии и обмен этими данными. Целью является создание механизма, позволяющего описывать данные об изделии на протяжении всего жизненного цикла изделия независимо от конкретной системы. Характер такого описания делает его пригодным не только для обмена инвариантными файлами, но также и для создания баз данных об изделиях, коллективного пользования этими базами и архивации соответствующих данных.

Стандарты серии ГОСТ Р ИСО 10303 представляют собой набор отдельно издаваемых стандартов (частей). Части данной серии стандартов относятся к одной из следующих тематических групп: методы описания, интегрированные ресурсы, прикладные протоколы, комплекты абстрактных тестов, методы реализации и аттестационное тестирование. Группы стандартов данной серии описаны в ГОСТ Р ИСО 10303-1. Настоящий стандарт входит в группу методов описания.

Настоящий стандарт определяет элементы языка EXPRESS-I. Каждый элемент языка представляется в своем собственном контексте с примерами. Сначала вводятся простые элементы, а затем представляются более комплексные идеи в порядке нарастания их сложности.

Обзор языка

EXPRESS-I – это название языка формального представления данных и спецификации абстрактных тестовых (контрольных) примеров. Он может быть использован для описания информационных требований других стандартов серии ГОСТ Р ИСО 10303 и родственен языкам EXPRESS и EXPRESS-G. Язык базируется на целом ряде целей проектирования, в частности:

- размеры и сложность стандартов серии ГОСТ Р ИСО 10303 требуют обеспечить читабельность языка как для компьютера, так и для человека. Выразительные средства стандартов данной серии призваны облегчить формальное выявление несоответствий в представлениях или спецификациях при использовании средств автоматизации;
- следует обратить внимание на отображение реализацией свойств объектов, представляющих предмет интереса. Обеспечить определение объекта в терминах его свойств, которые характеризуются установлением области их значений (домена) и ограничениями на эту область;
- обойтись, насколько это возможно, без рассмотрения конкретной реализации;
- обеспечить средства отображения малых совокупностей EXPRESS-схем;
- обеспечить средства поддержки спецификации комплектов абстрактных тестов для процессоров информационных моделей.

В EXPRESS-I экземпляры объектов представляются в терминах значений атрибутов: особенностей либо характеристик, считающихся важными для использования и понимания. Эти атрибуты имеют представление, которое может быть простым типом данных (таким, как целочисленный) либо типом другого объекта. Геометрическая точка может быть определена в терминах трех вещественных (действительных) чисел. Атрибутам, образующим определение объекта, даются имена. Так, для геометрической точки три вещественных числа могут иметь имена x, y и z. Устанавливается отношение между определяемым объектом и определяющими его атрибутами и, аналогичным образом, между атрибутом и его представлением.

Язык EXPRESS-I обеспечивает средства для отображения реализацией элементов данных языка EXPRESS. Язык разработан, в основном, для восприятия человеком и для облегчения отображения экземпляров EXPRESS-I на определении в EXPRESS-схеме. В некоторых стандартах серии ГОСТ Р ИСО 10303, например в ГОСТ Р ИСО 10303-21, установлены требования к рациональным машинным реализациям схем. EXPRESS-I не предназначен для замены этих методов.

Основные элементы языка показаны на рисунке 1. Язык имеет две главные части. Первая часть служит для отображения экземпляров данных. Данные могут быть отображены на основе объект—объект, на основе схемы либо как набор экземпляров схем, предназначенный для отображения некоторой информационной модели рассматриваемой предметной области. В языке EXPRESS-I эти данные называются экземплярами предмета (object instances), экземплярами данных схемы (schema data instances) и моделью (model). На рисунке 1 предполагается, что информационная модель определена с помощью языка EXPRESS.

Вторая часть языка служит для специфицирования абстрактных тестовых (контрольных) примеров с целью формального описания тестов, выполняемых над реализацией информационной модели, заданной на языке EXPRESS. Конструкциями языка, предназначенными для этой цели, являются контрольный пример (test case) и контекст (context). Данный раздел языка использует также процедурные аспекты языка EXPRESS. Экземпляры данных могут быть параметризованы и сохранены в контексте. Многие различные контрольные примеры могут присваивать значения параметризованным данным в контексте и использовать эти данные как часть спецификации данного теста.

Экземпляры данных, полученные в результате применения контрольного примера, могут отображаться с помощью конструкций, определенных в первой части языка.

Примечание — Примеры использования EXPRESS-I в настоящем стандарте не согласованы с правилами какого-либо конкретного стиля. В самом деле, иногда примеры используют не лучший стиль, чтобы достичь краткости либо показать гибкость. Примеры не претендуют на отражение содержания информационных моделей, определяемых в других стандартах серии ГОСТ Р ИСО 10303. Их функция — показать конкретные особенности EXPRESS-I. Любую аналогию между этими примерами и обязательными информационными моделями или абстрактными контрольными примерами, определенными в других стандартах серии ГОСТ Р ИСО 10303, следует игнорировать.

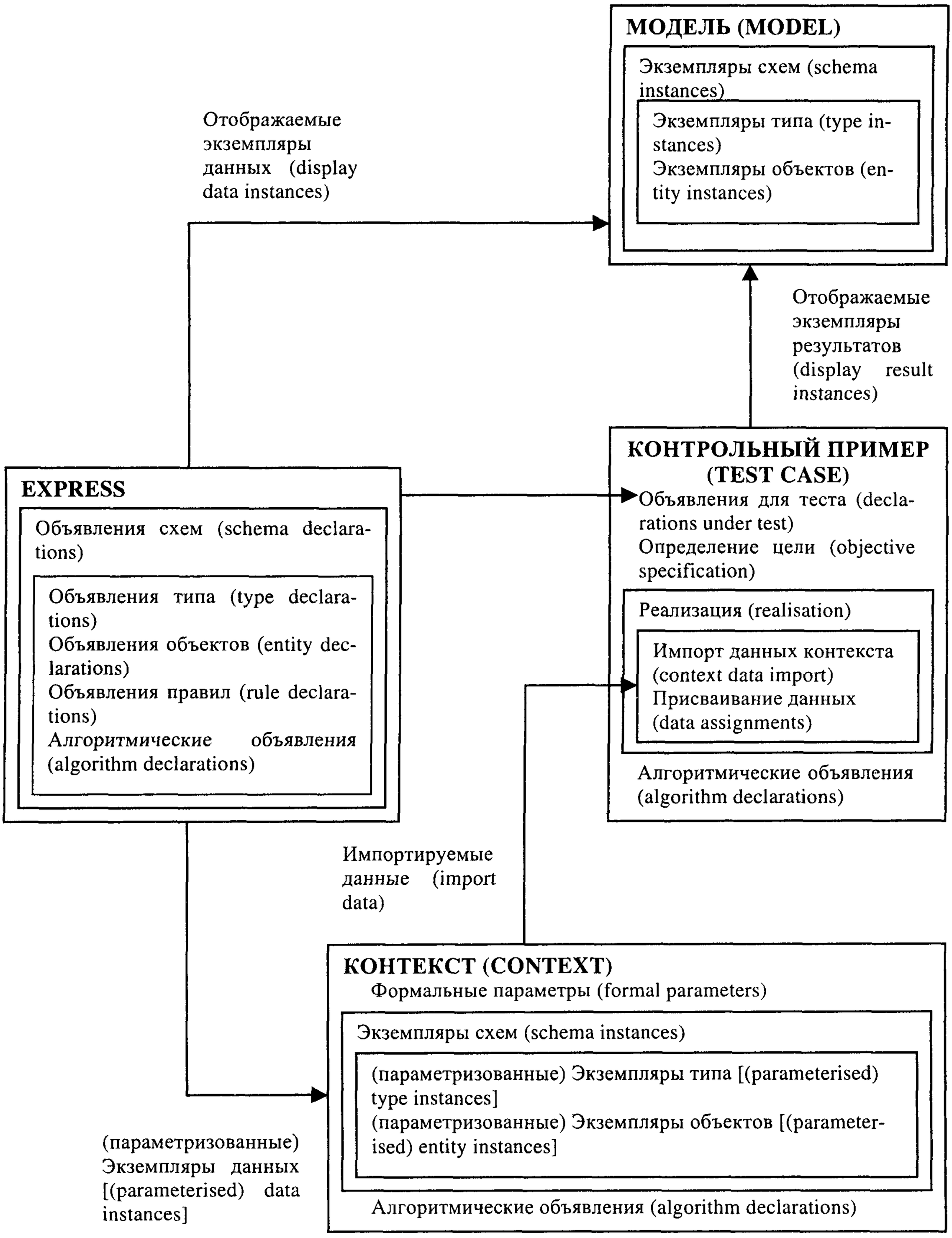


Рисунок 1 – Главные элементы языка EXPRESS-I

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Системы автоматизации производства и их интеграция

ПРЕДСТАВЛЕНИЕ ДАННЫХ ОБ ИЗДЕЛИИ И ОБМЕН ЭТИМИ ДАННЫМИ

Часть 12
Методы описания. Справочное руководство по языку EXPRESS-I

Industrial automation systems and integration. Product data representation and exchange.
Part 12. Description methods. The EXPRESS-I language reference manual

Дата введения 2002—01—01

1 Область применения

Настоящий стандарт определяет язык, на котором может быть описан (отображен) экземпляр (часть) рассматриваемой предметной области. Стандарт также определяет метод формального описания для поддержки спецификаций абстрактных тестовых (контрольных) примеров. Данный язык называется EXPRESS-I. Этот язык родственен языку EXPRESS, определенному в ГОСТ Р ИСО 10303-11.

EXPRESS-I является языком реализации для языка концептуальной схемы, как определено в ИСО/ТО 9007 [1], а конкретным языком концептуальной схемы, послужившим отправной точкой для EXPRESS-I, является EXPRESS. Язык EXPRESS-I позволяет отображать состояние предметов, принадлежащих к рассматриваемой предметной области, и блоки информации, относящиеся к этим предметам.

В область применения настоящего стандарта входят:
- отображение экземпляров схем;
- отображение экземпляров типов и объектов (сущностей);
- данные абстрактного тестового (контрольного) примера;
- преобразование EXPRESS-схем и типов данных в экземпляры EXPRESS-I.

В область применения настоящего стандарта не входят:
- преобразование из других языков (концептуальных схем) в EXPRESS-I;
- определение форматов базы данных;
- определение форматов файла;
- определение форматов передачи данных;
- управление процессом;
- обработка информации;
- обработка исключительных ситуаций.

EXPRESS-I не является языком программирования.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие стандарты:

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

ГОСТ Р ИСО 10303-11—2000 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 11. Методы описания. Справочное руководство по языку EXPRESS

ГОСТ Р ИСО 10303-21—99 Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 21. Методы реализации. Кодирование открытым текстом структуры обмена

Издание официальное

ИСО/МЭК 8824-1—95*) Информационная технология. Взаимосвязь открытых систем. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Спецификация основной нотации

ИСО 10303-31—94*) Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 31. Методология и основы аттестационного тестирования. Общие положения

ИСО/МЭК 10646-1—93*) Информационная технология. Универсальный многобайтно-кодированный набор символов. Часть 1. Архитектура и основной многоязычный уровень

3 Определения

3.1 Термины, определенные в ГОСТ Р ИСО 10303-1

В настоящем стандарте использованы следующие термины, определенные в ГОСТ Р ИСО 10303-1:

- **данные;**
- **информация;**
- **информационная модель.**

3.2 Термины, определенные в ГОСТ Р ИСО 10303-11

В настоящем стандарте использованы следующие термины, определенные в ГОСТ Р ИСО 10303-11:

- **граф подтип/супертип;**
- **значение;**
- **константа;**
- **лексема;**
- **объект;**
- **совокупность;**
- **тип данных сложного объекта;**
- **тип данных;**
- **экземпляр сложного объекта;**
- **экземпляр объекта;**
- **экземпляр простого объекта;**
- **экземпляр.**

3.3 Термины, определенные в ИСО 10303-31

В настоящем стандарте использованы следующие термины, определенные в ИСО 10303-31:

- **абстрактный тестовый (контрольный) пример;**
- **критерий вердикта;**
- **цель теста.**

3.4 Другие определения

В настоящем стандарте использованы следующие термины с соответствующими определениями:

3.4.1 **атрибут** (attribute): Особенность, качество или свойство, характеризующее объект.

3.4.2 **информационная база** (information base): Набор экземпляров типов, соответствующих друг другу и информационной модели, принадлежащий экземпляру рассматриваемой предметной области.

Примечание – Информационная база может либо не может быть пригодной для компьютерной обработки. Например, ее не следует считать пригодной для компьютерной обработки, если она имеет форму рукописного документа. С другой стороны, если она задана в виде базы данных или компьютерного файла, то ее следует считать пригодной для компьютерной обработки и, следовательно, ее можно также называть объектной базой.

3.4.3 **объектная база** (object base): Информационная база, пригодная для компьютерной обработки.

3.4.4 **схема** (schema): Набор тесно связанных элементов, образующий информационную модель либо ее часть.

3.4.5 **тип** (type): Представление области (домена) допустимых значений.

---

*) Оригиналы стандартов ИСО (ИСО/МЭК) – во ВНИИКИ Госстандарта России.

3.4.6 **рассматриваемая предметная область** (universe of discourse): Все те объекты (предметы) реального мира, которые представляют потенциальный интерес. Они являются подмножеством всех предметов (объектов) реального мира.

4 Требования соответствия

4.1 Формальные спецификации, записанные на EXPRESS-I

Формальная спецификация, записанная на EXPRESS-I, должна быть согласована с заданным уровнем соответствия, как определено ниже. Формальная спецификация соответствует данному уровню, когда для нее успешно выполнены все проверки, предусмотренные для данного и всех нижележащих уровней.

4.1.1 Уровни соответствия

Уровень 1. Проверка ссылок.

Этот уровень включает проверку формальной спецификации на корректность синтаксиса и ссылок. Формальная спецификация синтаксиса верна, если она соответствует синтаксису, образованному расширением первичных синтаксических правил, приведенных в приложении А. Формальная спецификация верна по ссылкам, если все ссылки на элементы EXPRESS-I соответствуют области действия и правилам видимости, определенным в разделе 11.

Уровень 2. Проверка типов.

Данный уровень включает проверку формальной спецификации на совместимость типов в выражениях и операторах присваивания, как определено для проверок уровня 2 в ГОСТ Р ИСО 10303-11.

Уровень 3. Проверка значений.

Данный уровень включает проверку формальной спецификации на соответствие условиям, определяемым проверками уровня 3 по ГОСТ Р ИСО 10303-11.

Уровень 4. Полная проверка.

Данный уровень включает проверку формальной спецификации на соответствие всем формулировкам требований, установленным в настоящем стандарте.

4.2 Реализации EXPRESS-I

Реализация синтаксического анализатора языка EXPRESS-I должна обеспечивать синтаксический разбор любой формальной спецификации, записанной на EXPRESS-I, в соответствии с ограничениями, связанными с данной реализацией, которые определены в ЗСРП (приложение В). Синтаксический анализатор языка EXPRESS-I должен быть продекларирован на соответствие конкретному уровню (как это определено в 4.1.1), если он может реализовать все проверки, предусмотренные для данного уровня (и всех нижележащих уровней) для формальной спецификации, записанной на EXPRESS-I.

Разработчик синтаксического анализатора языка EXPRESS-I должен указать все ограничения, которая реализация накладывает на число и длину идентификаторов, диапазон обрабатываемых чисел и максимальную точность вещественных чисел. Такие ограничения должны документироваться для аттестационного тестирования в виде, определенном в приложении В.

5 Основные принципы

Предполагается, что читатель настоящего стандарта знаком с языком EXPRESS, описанным в ГОСТ Р ИСО 10303-11.

Использование EXPRESS-I для отображения экземпляров не требует и не предполагает наличия соответствующего набора объектов или других определений. Это означает, что EXPRESS-I может использоваться как язык со своими собственными правилами. Однако обычно вводится сопутствующий набор определений, описанных формальным образом на основе языка типа EXPRESS.

EXPRESS-I не описывает среду реализации. В частности, EXPRESS-I не определяет:

- как организован доступ или вывод данных экземпляра;
- как организовано хранение или обслуживание данных экземпляра;
- как разрешены ссылки на EXPRESS-схемы;
- как и когда проверяются ограничения или о них сообщается.

6 Элементы языка

В данном разделе установлены основные элементы, из которых формируют предложения языка EXPRESS-I: набор символов, примечания, знаки, зарезервированные слова и идентификаторы.

Определения синтаксиса, приведенные в настоящем стандарте в прямоугольных рамках, являются выдержками из синтаксиса языка EXPRESS-I, установленного в приложении А, которое определяет полный синтаксис языка и любые языковые конструкции, не представленные в настоящем разделе. Метод спецификации синтаксиса является расширением множества, используемого для EXPRESS в соответствии с разделом 6 ГОСТ Р ИСО 10303-11.

Примечание 1 — Для удобства читателя метод определения EXPRESS повторен в приложении D вместе с расширениями для EXPRESS-I.

Базовые элементы языка компонуются в поток исходного текста, обычно разделяемого на физические строки. Физической строкой является любое число символов (включая ноль), заканчивающееся символом новой строки (см. 6.1.5.2).

Примечание 2 — Исходный текст EXPRESS-I легче читать, если операторы представлены отдельными строками, а для разделения различных конструкций использованы пробелы.

6.1 Набор символов

В исходном тексте EXPRESS-I следует использовать только символы из следующего набора: символы, размещенные в ячейках 20—7E ряда 00 уровня 00 группы 00 ИСО/МЭК 10646-1 и специальный символ \n, обозначающий новую строку. Этот набор называется набором символов EXPRESS-I. Элементы этого набора ссылаются на соответствующие ячейки ИСО/МЭК 10646-1, в которых определены данные символы; номера этих ячеек определены в шестнадцатеричной системе. Печатаемые символы данного набора (ячейки 21—7E из ИСО/МЭК 10646-1) комбинируются для образования лексем языка EXPRESS-I. Лексемами EXPRESS-I являются ключевые слова, идентификаторы, знаки, литералы или значения. Дальнейшая классификация набора символов EXPRESS-I приведена ниже.

Определенный таким образом набор символов является абстрактным набором символов; он не зависит от его представления в реализации. В частности, фактическая реализация может использовать некоторые управляющие коды, определенные в ИСО/МЭК 6429 [2]. Такие коды интерпретируются реализацией и в результате могут приводить к включению в окончательный исходный текст одного или нескольких абстрактных символов из набора символов EXPRESS-I.

Пример 1 — Управляющий код TAB может интерпретироваться реализацией как добавление одного или нескольких символов пробела к абстрактным символам, образующим определение EXPRESS-I.

Примечание — Данный раздел определяет только символы, используемые для определения исходного текста на EXPRESS-I, но не определяет область символов, допустимых внутри строкового значения.

6.1.1 Цифры

В EXPRESS-I используются арабские цифры 0—9 (ячейки 30—39 набора символов EXPRESS-I).

Синтаксис:
120 digit = < как в EXPRESS > .

6.1.2 Буквы

В EXPRESS-I используются прописные и строчные буквы английского алфавита (ячейки 41—5A и 61—7A набора символов EXPRESS-I). Тип букв имеет значение только внутри явных строковых значений.

Примечание — EXPRESS-I может быть описан с использованием прописных, строчных или и тех, и других букв.

Синтаксис:
124 letter = < как в EXPRESS > .

6.1.3 Специальные символы

Специальные символы (печатаемые символы, не являющиеся буквами или цифрами) используются, в основном, для пунктуации и в качестве операторов. Некоторые из указанных спе-

циальных символов не используются как часть языка. Они, однако, могут использоваться внутри примечаний и строковых значений. Такие специальные символы находятся в ячейках 21—2F, 3A—3F, 40, 5B—5E, 60 и 7B—7E набора символов EXPRESS-I.

Синтаксис:
134 special = < как в EXPRESS > .

6.1.4 Подчеркивание

Символ подчеркивания (_ , ячейка 5F набора символов EXPRESS-I) может быть использован в идентификаторах и ключевых словах, но он не должен использоваться в качестве первого символа.

6.1.5 Пробел

Пробел определяется в следующих подпунктах и в 6.1.6. Пробел должен использоваться для разделения лексем в исходном тексте EXPRESS-I.

Примечание — Свободное, в рамках допустимого, использование пробелов может улучшить структуру и читабельность исходного текста на EXPRESS-I.

6.1.5.1 *Символ пробела*

Один или несколько пробелов (ячейка 20 набора символов EXPRESS-I) могут появиться между двумя лексемами или внутри строкового значения. Обозначение \s можно использовать для представления символа пробела в синтаксисе языка.

6.1.5.2 *Новая строка*

Новая строка помечает физический конец строки внутри формальной спецификации, записанной на EXPRESS-I. Новая строка обычно выступает как пробел, но требуется и по существу, когда она завершает концевое примечание или появляется внутри строкового значения. Новая строка представляется обозначением \n в синтаксисе языка.

Представление новой строки определяется реализацией.

6.1.6 Примечания

Примечание используется для документирования и должно интерпретироваться синтаксическим анализатором EXPRESS-I как пробел. Имеются две формы примечания: встроенное и концевое.

6.1.6.1 *Встроенное примечание*

Пара символов (* обозначает начало встроенного примечания, а пара символов *) обозначает его конец. Встроенное примечание может появляться между любыми двумя лексемами.

Синтаксис:
142 embedded_remark = < как в EXPRESS > .

Любой символ из набора символов EXPRESS-I может находиться между началом и концом встроенного примечания, включая символ новой строки; поэтому встроенные примечания могут занимать несколько физических строк.

Встроенные примечания могут быть вложенными.

Примечание — Следует следить за тем, чтобы вложенные примечания обрамлялись парами соответствующих знаков.

Пример 2 — Пример встроенных вложенных примечаний.

```
(* Знак `(*` начинает встроенное примечание, а знак `*)` заканчивает его. *)
```

6.1.6.2 *Концевое примечание*

Концевое примечание записывается в конце физической строки. Два последовательных дефиса (--) служат началом концевого примечания, а последующий символ новой строки завершает его.

Синтаксис:
144 tail_remark = <как в EXPRESS> .

Пример 3 — Концевое примечание.

```
-- Это концевое примечание, и оно завершается символом новой строки.
```

6.2 Зарезервированные слова

Зарезервированными словами в EXPRESS-I являются ключевые слова и имена встроенных констант, функций и процедур. Зарезервированные слова не должны использоваться в качестве идентификаторов. Зарезервированные слова EXPRESS-I описаны ниже.

6.2.1 Ключевые слова

В EXPRESS-I используется подмножество ключевых слов EXPRESS вместе с некоторыми дополнительными ключевыми словами.

В таблице 1 перечислены ключевые слова, общие для EXPRESS-I и EXPRESS. В таблице 2 перечислены дополнительные ключевые слова EXPRESS-I.

Примечание – Ключевые слова обозначаются прописными буквами, представляющими литерал. Это позволяет облегчить чтение синтаксических конструкций.

Таблица 1 – Ключевые слова, общие для EXPRESS-I и EXPRESS

ABSTRACT	AGGREGATE	ALIAS	ARRAY
BAG	BEGIN	BINARY	BOOLEAN
BY	CASE	CONSTANT	CONTEXT
DERIVE	ELSE	END	END_ALIAS
END_CASE	END_CONSTANT	END_CONTEXT	END_ENTITY
END_FUNCTION	END_IF	END_LOCAL	END_MODEL
END_PROCEDURE	END_REPEAT	END_TYPE	ENTITY
ENUMERATION	ESCAPE	FIXED	FOR
FUNCTION	GENERIC	IF	INTEGER
INVERSE	LIST	LOCAL	LOGICAL
MODEL	NUMBER	OF	ONEOF
OPTIONAL	OTHERWISE	PROCEDURE	QUERY
REAL	REPEAT	RETURN	SELECT
SET	SKIP	STRING	SUBTYPE
SUPERTYPE	THEN	TO	TYPE
UNIQUE	UNTIL	VAR	WHERE
WHILE			

Таблица 2 – Дополнительные ключевые слова EXPRESS-I

CALL	CRITERIA	END_CALL	END_CRITERIA
END_NOTES	END_OBJECTIVE	END_PARAMETER	END_PURPOSE
END_REALIZATION	END_REFERENCES	END_SCHEMA_DATA	END_TEST_CASE
IMPORT	NOTES	OBJECTIVE	PARAMETER
PURPOSE	REALIZATION	REFERENCES	SCHEMA_DATA
SUBOF	SUPOF	TEST_CASE	USING
WITH			

6.2.2 Зарезервированные слова, являющиеся операторами

Операторы, определяемые зарезервированными словами, указаны в таблице 3. Это те же самые операторы, что и операторы EXPRESS, определенные в разделе 12 ГОСТ Р ИСО 10303-11.

Таблица 3 – Использование в EXPRESS-I операторов EXPRESS

AND	ANDOR	DIV	IN
LIKE	MOD	NOT	OR
XOR			

6.2.3 Встроенные константы

Имена встроенных констант EXPRESS-I приведены в таблице 4. Это те же константы, что и константы EXPRESS, определенные в разделе 14 ГОСТ Р ИСО 10303-11.

Таблица 4 – Использование в EXPRESS-I констант EXPRESS

?	CONST_E	FALSE	PI
SELF	TRUE	UNKNOWN	

6.2.4 Встроенные функции

Имена функций EXPRESS, которые могут использоваться в EXPRESS-I, приведены в таблице 5.

Таблица 5 – Использование в EXPRESS-I функций EXPRESS

ABS	ACOS	ASIN	ATAN
BLENGTH	COS	EXISTS	EXP
FORMAT	HIBOUND	HIINDEX	LENGTH
LOBOUND	LOG	LOG10	LOG2
LOINDEX	NVL	ODD	ROLESOF
SIN	SIZEOF	SQRT	TAN
TYPEOF	USEDIN	VALUE	VALUE_IN
VALUE_UNIQUE			

Определения этих функций приведены в разделе 15 ГОСТ Р ИСО 10303-11.

6.2.5 Встроенные процедуры

Имена EXPRESS-процедур, которые могут быть использованы в EXPRESS-I, приведены в таблице 6. Процедуры определены в разделе 16 ГОСТ Р ИСО 10303-11.

Таблица 6 – Использование в EXPRESS-I процедур EXPRESS

INSERT	REMOVE

6.3 Знаки

Знаками являются специальные символы или группы специальных символов, имеющие специфический смысл в EXPRESS-I. Знаки используются в EXPRESS-I в качестве ограничителей и операторов. Ограничитель используется для начала, разделения или завершения соседних лексических или синтаксических элементов. Интерпретация этих элементов была бы невозможной без разделителей. Операторы означают, что над операндами должны быть совершены действия, связанные с оператором. Знаки EXPRESS-I приведены в таблицах 7 и 8.

Таблица 7 – Знаки, общие для EXPRESS-I и EXPRESS

.	,	;	:
*	+	-	=
%	'	\	/
<	>	[	]
{	}	|	e
(	)	<=	< >
>=	<*	:=	||
**	- -	(*	*)
:=:	:<>:		

Таблица 8 – *Дополнительные знаки* EXPRESS-I

@	!	->	<-
= =	“		

6.4 Идентификаторы и ссылки

Идентификаторами являются имена, присвоенные элементам, объявленным в реализации EXPRESS-I. Идентификатор не должен совпадать с зарезервированными словами EXPRESS-I или EXPRESS.

Синтаксис:

```
187 constant_id = < как в EXPRESS > .
198 entity_id = < как в EXPRESS > .
282 schema_id = < как в EXPRESS > .
140 simple_id = < как в EXPRESS > .
51i ComplexEntityInstanceId = SimpleEntityInstanceId `[` SupSubId `]` .
58i ContextId = simple_id .
69i EntityInstanceId = ComplexEntityInstanceId |
                       SimpleEntityInstanceId .
73i EnumerationId = type_ref .
75i EnumerationInstanceId = simple_id .
92i ModelId = simple_id .
100i ParameterId = simple_id .
115i SelectId = type_ref .
117i SelectInstanceId = simple_id .
120i SimpleEntityInstanceId = simple_id .
122i SimpleInstanceId = simple_id .
125i SupSubId = digits .
129i TestCaseId = simple_id .
136i TypeId = type_ref .
138i TypeInstanceId = simple_id .
```

Первым символом простого идентификатора должна быть буква. Остальные символы, при их наличии, могут быть любой комбинацией букв, цифр и символа подчеркивания. Внутри идентификаторов не должно быть ни одного пробела.

Разработчик синтаксического анализатора EXPRESS-I должен установить максимальное число символов идентификатора, которое может распознаваться данной реализацией (см. приложение В).

Примечание – Буквы, используемые для формирования идентификаторов, не чувствительны к типу, т.е. заглавные и строчные буквы воспринимаются как одинаковые.

Пример 4 – Правильные простые идентификаторы

POINT line Circle AnEntity item507 An_integer.

Пример 5 – Неправильные простые идентификаторы

_POINT	подчеркивание не может быть первым символом
line?	? не может быть частью идентификатора
3dThing	цифра не может быть первым символом
Pi	Pi является ключевым словом EXPRESS-I

Пример 6 – Правильные идентификаторы экземпляра сложного объекта

complex[101] complex[12] an_ent[23] an_ent[77]

Синтаксис:

```
146 constant_ref = < как в EXPRESS > .
154 type_ref = < как в EXPRESS > .
36i ContextRef = ContextId .
39i ParameterRef = ParameterId .
```

На элемент можно ссылаться по его идентификатору. На элементы — константа и параметр — можно ссылаться по соответствующему идентификатору.

```
Синтаксис:
34i  ComplexEntityInstanceRef = `@` SipmleEntityInstanceId  .
37i  EntityInstanceRef = ComplexEntityInstanceRef |
                         SimpleEntityInstanceRef  .
38i  EnumerationInstanceRef = `@` EnumerationInstanceId  .
96i  ObjectInstanceRef = EntityInstanceRef | EnumerationInstanceRef |
                         SelectInstanceRef | SimpleInstanceRef |
                         TypeInstanceRef  .
40i  SelectInstanceRef = `@` SelectInstanceId  .
41i  SimpleEntityInstanceRef = `@` SimpleEntityInstanceId  .
42i  SimpleInstanceRef = `@` SimpleInstanceId  .
43i  SupSubRef = `@` SupSubId .
44i  TypeInstanceRef = `@` TypeInstanceId  .
```

Первым символом ссылки на экземпляр объекта (entity), перечисления (enumeration), типа (type) или выбора (select) должен быть @ с последующим хотя бы одним символом. Символы после начального @ могут быть любой комбинацией букв, цифр и символа подчеркивания, которые образуют правильный идентификатор объекта, перечисления, простого экземпляра, экземпляра выбора или типа. Эти ссылки в совокупности называются ссылками на экземпляр предмета.

Пример 7 — Правильные ссылки на экземпляр предмета

@POINT @line @Circle @AnEntity @item567

Пример 8 — Неправильные ссылки на экземпляр предмета.

@line?	? не может быть частью идентификатора
3dThing	@ должен быть первым символом
@subof	subof является ключевым словом EXPRESS-I
@@Circle	@ может появляться только в качестве первого символа
@567	символы, следующие за @, должны начинаться с буквы
@complex[82]	допустимы только символы букв, цифр и подчеркивания.

7 Именованные области значений

В настоящем разделе определены типы областей значений (доменов), как части языка. Области значений используются для описания допустимых значений экземпляра. Именованными областями значений являются области значений объекта, типа, перечисления или выбора.

7.1 Область значений объекта

Область значений объекта представляет класс предметов, имеющих общие атрибуты.

```
Синтаксис:
66i  EntityDomain = [ SchemaId `.` ] EntityId  .
```

Примечание — Область значений объекта соответствует типу данных объекта в языке EXPRESS.

7.2 Область значений перечисления

Область значений перечисления охватывает область значений упорядоченное множество имен.

```
Синтаксис:
72i  EnumerationDomain = [ SchemaId `.` ] EnumerationId  .
```

Примечание — Область значений перечисления соответствует перечисляемому типу данных в языке EXPRESS.

7.3 Область значений выбора

Область значений выбора охватывает объединение областей значений.

Синтаксис:
114i SelectDomain = [SchemaId `.`] SelectId .

Примечание – Область значений выбора соответствует выбираемому типу данных в языке EXPRESS.

7.4 Область значений типа

Область значений типа является расширением для других областей значений в языке.

Синтаксис:
135i TypeDomain = [SchemaId `.`] TypeId.

Примечание – Область значений типа соответствует определенному типу данных в языке EXPRESS, не являющемуся перечисляемым или выбираемым типом.

8 Значения и экземпляры

В данном разделе описаны реализации возможностей языка EXPRESS.

8.1 Базовые значения

Синтаксис:
48i BaseValue = SimpleValue | EnumerationValue .
123i SimpleValue = BinaryValue | BooleanValue | LogicalValue |
NumberValue | StringValue .

Простым значением является значение самоопределенной константы. Область значения зависит от того, как компонуются символы при формировании лексемы.

8.1.1 Двоичное значение

Двоичное значение представляет величину двоичной области значений.

Синтаксис:
25i BinaryValue = binary_literal .
136 binary_literal = < как в EXPRESS > .

Двоичное значение образуется из символа % и последующих одного или более битов (0 или 1).

Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальное число битов в двоичном значении, которое может распознаваться данной реализацией (см. приложение В).

Пример 9 – Правильное двоичное значение

%101001100000101

8.1.2 Булевское значение

Булевское значение представляет величину булевской области значений.

Синтаксис:
50i BooleanValue = TRUE | FALSE .

Булевским значением является одна из встроенных констант FALSE или TRUE.

8.1.3 Числовое значение

Числовым значением является целочисленное или действительное значение.

Синтаксис:
94i NumberValue = IntegerValue | RealValue .

8.1.4 Целочисленное значение

Целочисленное значение представляет величину целочисленной области значений.

Синтаксис:
29i IntegerValue = [sign] integer_literal .
138 integer_literal = < как в EXPRESS > .
286 sign = < как в EXPRESS > .

Целочисленный литерал компонуется полностью из цифр. Целочисленное значение компонуется из целочисленного литерала, возможно, предваряемого знаком. Оно определяет положительное, отрицательное либо нулевое целое число.

Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальное значение целого, которое может распознаваться данной реализацией (см. приложение В).

Пример 10 – Правильные целочисленные значения:

0 1 -1 891562934527619

Пример 11 – Неправильные целочисленные значения:

1.0 не может включать десятичную точку.

8.1.5 Логическое значение

Логическое значение представляет величину логической области значений.

```
Синтаксис
88i  LogicalValue = logical_literal  .
242  logical_literal = < как в EXPRESS >  .
```

Логическим значением является одна из встроенных констант FALSE, TRUE или UNKNOWN.

8.1.6 Действительное значение

Действительное (вещественное) значение представляет величину действительной области значений.

Действительным значением является математическая константа со знаком или действительный литерал со знаком.

```
Синтаксис:
104i RealValue = SignedMathConstant | SignedRealLiteral  .
31i  SignedMathConstant = [ sign ] MathConstant  .
89i  MathConstant = CONST_E | PI  .
32i  SignedRealLiteral = [ sign ]  real_literal  .
139  real_literal = < как в EXPRESS >  .
```

Математической константой со знаком является одна из встроенных математических констант (то есть *e* или π), возможно предваряемая знаком.

Математическая константа $e = 2{,}7182...$ представляется константой CONST_E языка EXPRESS.

Математическая константа $\pi = 3{,}1415...$ представляется константой PI языка EXPRESS.

Пример 12 – Математические константы со знаком:

-const_e Pi

Действительный литерал со знаком компонуется из мантиссы (со знаком) и возможного показателя степени. Он определяет действительное число.

Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальную точность и максимальный показатель степени действительного значения, которые могут распознаваться данной реализацией (см. приложение В).

Пример 13 – Правильные действительные значения:

0.0 -1.E6 1.e-6 8915629.34527619

Пример 14 – Неправильные действительные значения:

.001 перед точкой должна быть хотя бы одна цифра
1e10 в мантиссе должна быть десятичная точка
1.0 e-12.0 в показателе не должно быть десятичной точки
CONSTE неверно записанная встроенная константа.

8.1.7 Строковое значение

Строковое значение представляет величину строковой области значений. Имеются две формы строкового значения – явное строковое значение и кодированное строковое значение. Явное

строковое значение компонуется из последовательности символов из набора символов EXPRESS-I, заключенной в апострофы ('). Апостроф внутри явного строкового значения представляется двумя последовательными апострофами. Кодированным строковым значением является кодированное четырьмя октетами представление последовательности символов из ИСО/МЭК 10646-1, заключенное в кавычки (“). Кодирование определяется следующим образом:

- первый октет = группа ИСО/МЭК 10646-1, в которой определяется символ;
- второй октет = проекция ИСО/МЭК 10646-1, в которой определяется символ;
- третий октет = строка ИСО/МЭК 10646-1, в которой определяется символ;
- четвертый октет = ячейка ИСО/МЭК 10646-1, в которой определяется символ.

Последовательность октетов должна идентифицировать один из допустимых символов из ИСО/МЭК 10646-1.

```
Синтаксис:
124i StringValue = SimpleStringValue | EncodedStringValue .
33i  SimpleStringValue = \q { (\q \q) | not_quote | \s | \o | \n } \q .
130  not_quote = < как в EXPRESS > .
27i  EncodedStringValue = ' “ ' { encoded_character | \n } ' “ ' .
122  encoded_character = < как в EXPRESS > .
```

Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальное число символов строкового значения, которое может распознаваться данной реализацией (см. приложение В).

Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальное число октетов (должно быть кратно четырем) кодированного строкового значения, которое может распознаваться данной реализацией (см. приложение В).

Примечание – Строковое значение EXPRESS отличается от строкового литерала EXPRESS, поскольку в первом случае строковое значение может занимать более чем одну физическую строку, тогда как строковой литерал EXPRESS не может занимать более одной физической строки.

Пример 15 – Правильные явные строковые значения:

‘This is a string on the line.’
Читается: это строка в одну физическую строку.
‘This
is
a
multiline
string’.
Читается: Это
многострочная
строка
‘This string’‘s got a single apostrophe embedded in it’.
Читается: Эта строка содержит единственный встроенный апостроф.

Пример 16 – Неправильные явные строковые значения

‘This string is invalid because there is no closing apostrophe.

Пример 17 – Правильные кодированные строковые значения

“00000041”
читается: A.
“000000C5“
читается: Å

Пример 18 – Неправильные кодированные строковые значения:

“000041“
Октеты должны раскладываться в группы четверок.
“00000041 000000C5“
Между октетами не должно быть пробелов.

8.1.8 Перечисляемое значение

Перечисляемое значение представляет величину перечисляемой области значений.

Синтаксис:
28i EnumerationValue = `!` simple_id .

Перечисляемое значение является простым идентификатором с предшествующим восклицательным знаком (!). Простым идентификатором является последовательность символов из букв, цифр и символа подчеркивания с буквой в качестве первого символа.

Пример 19 – Правильные перечисляемые значения:

!red !green !forward

8.2 Агрегатные значения

В EXPRESS различают две формы значений агрегаций – фиксированную и динамическую. Фиксированная агрегация является агрегацией аналогичных предметов, где число мест хранения не зависит от количества элементов, фактически хранящихся в агрегации. Динамическая агрегация является агрегацией аналогичных предметов, где число мест хранения зависит от числа элементов, фактически хранящихся в агрегации. Агрегатные значения могут быть вложенными.

Синтаксис:
46i AggregationValue = DynamicAggr | FixedAggr .
61i DynamicAggr = `(` [DynamicList] `)` .
63i DynamicList = DynamicMember { `,` DynamicMember } .
64i DynamicMember = AggregationValue | ConstantValue | DerattValue | ParmValue | RegattValue | TypeValue .
79i FixedAggr = `[` FixedList `]` .
80i FixedList = FixedMember { `,` FixedMember } .
81i FixedMember = DynamicMember | Nil .

Допустимые области значений элементов внутри агрегации зависят от контекста области значений. Такими контекстами являются:

- константы (см. 8.8);
- вычисляемые атрибуты (см. 8.7.1.2);
- явные атрибуты (см. 9.2.2);
- параметры (см. 9.2.2);
- определенные типы данных (см. 8.4).

Правила и ограничения

a) Элементы внутри динамической агрегации не должны быть равны Nil.

b) Элементы внутри фиксированной агрегации могут быть равны Nil.

c) Значения элементов внутри агрегации должны быть совместимы с областью значений агрегации.

Пример 20 – Агрегатные значения:

(10, -10, 0)	динамическая агрегация трех целочисленных значений
(1, 1, 2, 2, 3, 3)	динамическая агрегация шести целочисленных значений
( )	пустая динамическая агрегация
[1, 2, 3, 4]	фиксированная агрегация четырех целочисленных значений
([1, 2],[3, ?])	динамическая агрегация фиксированной агрегации двух значений.

8.3 Простой экземпляр

Простой экземпляр является представлением значения одного экземпляра простого значения.

Синтаксис:
121i SimpleInstance = SimpleInstanceId `=` SimpleValue `;` .
122i SimpleInstanceId = simple_id .
123i SimpleValue = BinaryValue | BooleanValue | LogicalValue | NumberValue | StringValue .
42i SimpleInstanceRef = `@` SimpleInstanceId .

Пример 21 – Некоторые простые экземпляры

r1 = 27.0;
&1 = ' A string';

8.4 Экземпляр типа

Экземпляр типа является представлением значения одного экземпляра области значений типа (TYPE).

```
Синтаксис:
137i  TypeInstance = TypeInstanceId `=` TypeInstanceValue `;` .
138i  TypeInstanceId = simple_id .
139i  TypeInstanceValue = TypeDomain `{` TypeValue `}` .
140i  TypeValue = AggregationValue | BaseValue | ConstantRef |
                  EntityInstanceValue | NamedInstanceValue |
                  ObjectInstanceRef | ParameterRef .
44i   TypeInstanceRef = `@` TypeInstanceId .
```

Правила и ограничения

а) Значение экземпляра должно быть либо простым значением, ссылкой на экземпляр объекта, ссылкой на экземпляр типа, либо агрегациями этих значений.

Пример 22 – Некоторые экземпляры типов:

t1 = a_real{27.0};
t2 = an_array_of_string {[`one`, `two`]};
t3 = a_dynamic_aggregate_of_integer {(1, 1, 2, 3, 5, 8, 13)};

8.5 Экземпляр выбора

Экземпляр выбора является представлением значения одного экземпляра области значений выбора (SELECT).

```
Синтаксис:
116i  SelectInstance = SelectInstanceId `=` SelectInstanceValue `;` .
117i  SelectInstanceId = simple_id .
118i  SelectInstanceValue = SelectDomain `{` SelectValue `}` .
119i SelectValue = EnumerationValue | NamedInstanceValue | ObjectInstanceRef |
                   TypeValue .
40i   SelectInstanceRef = `@` SelectInstanceId .
```

Правила и ограничения

а) Значение экземпляра должно быть либо ссылкой на экземпляр типа, ссылкой на экземпляр выбора, ссылкой на экземпляр перечисления, либо ссылкой на экземпляр объекта.

Пример 23 – Экземпляр выбора

&1 = type_or_entity{@e27};

8.6 Экземпляр перечисления

Экземпляр перечисления является представлением значения одного экземпляра области значений перечисления (ENUMERATION).

```
Синтаксис:
74i  EnumerationInstance = EnumerationInstanceId `=`
                           EnumerationInstanceValue `;` .
75i  EnumerationInstanceId = simple_id .
76i  EnumerationInstanceValue = EnumerationDamain
                                `{` EnumerationValue `}` .
28i  EnumerationValue = `!` simple_id .
38i  EnumerationInstanceRef = `@` EnumerationInstanceId .
```

Правила и ограничения

а) Значение экземпляра должно быть перечисляемым значением.

Пример 24 – Некоторые экземпляры перечисления

```
enum1 = an_enum{!first};
enum2 = an_enum{!second};
```

8.7 Экземпляр объекта

Экземпляр объекта является представлением одного экземпляра области значений объекта (ENTITY).

```
Синтаксис:
68i  EntityInstance = EntityInstanceId `=` EntityInstanceValue `;` .
69i  EntityInstanceId = ComplexEntityInstanceId |
                        SimpleEntityInstanceId .
70i EntityInstanceValue = EntityDomain `{` [ InheritsFrom ]
                          { ExplicitAttr } { DerivedAttr }
                          { InverseAttr } [ BequeathesTo ] `}` .
37i  EntityInstanceRef = ComplexEntityInstanceRef |
                         SimpleEntityInstanceRef .
```

В EXPRESS различают два вида экземпляра объекта:

- **экземпляр простого объекта** - экземпляр не являющийся частью дерева наследования;
- **экземпляр сложного объекта** - экземпляр из дерева наследования. Он компонуется из экземпляров компонентов (объектов), которые вместе образуют все узлы дерева.

```
Синтаксис:
51i  ComplexEntityInstanceId = SimpleEntityInstanceId `[` SupSubId `]` .
34i  ComplexEntityInstanceRef = `@` SimpleEntityInstanceId .
120i SimpleEntityInstanceId = simple_id .
41i  SimpleEntityInstanceRef = `@` SimpleEntityInstanceId .
125i SupSubId = digits .
```

Идентификатором экземпляра простого объекта является простой идентификатор.

Идентификатор экземпляра сложного объекта состоит из двух частей. Первая часть является такой же как идентификатор экземпляра простого объекта. Второй частью является строка цифр, заключенная в квадратные скобки. Строка цифр во второй части (называемая в синтаксисе SupSubId) является идентификатором конкретного компонента экземпляра сложного объекта. Ссылка на экземпляр сложного объекта включает первую часть идентификатора с предшествующим ему символом @.

Правила и ограничения

a) Для данного экземпляра сложного объекта первая часть идентификатора экземпляра сложного объекта должна быть такой же, что и для каждого компонента экземпляра сложного объекта.

b) *Для данного экземпляра сложного объекта вторая часть идентификатора экземпляра* сложного объекта должна быть разной для каждого компонента экземпляра сложного объекта.

Пример 25 – Идентификатор экземпляра сложного объекта для двухкомпонентного экземпляра и ссылка на этот экземпляр сложного объекта.

```
complex[23]      - - идентификатор одного компонента
complex[111]     - - идентификатор другого компонента
@complex         - - ссылка на экземпляр сложного объекта
```

8.7.1 Атрибуты

Экземпляр объекта в EXPRESS может не иметь ни одного (ноль) или иметь несколько атрибутов. Атрибуты подразделяются на явные, вычисляемые и инверсные.

Пример 26 – Экземпляры пустых объектов

```
e2 = ent_inst{ };
eg = ent_inst{ };
```

8.7.1.1 *Явные атрибуты*

Явный атрибут является обязательным свойством объекта.

```
Синтаксис:
77i   ExplicitAttr = RequiredAttr | OptionalAttr .
106i RequiredAttr = RoleName `->` (ReqattValue | Nil) `;` .
99i   OptionalAttr = RoleName `->` OptattValue `;` .
107i RoleName = attribute_ref .
105i ReqattValue = AggregationValue | BaseValue | ConstantRef |
                   NamedInstanceValue | ObjectInstanceRef |
                   ParameterRef | SelectValue | TypeValue .
96i   ObjectInstanceRef = EntityInstanceRef | EnumerationInstanceRef |
                          SelectInstanceRef | TypeInstanceRef |
                          SimpleInstanceRef .
93i   NamedInstanceValue = EnumerationInstanceValue | SelectInstanceValue |
                           TypeInstanceValue .
98i   OptattValue = ReqattValue | Nil .
30i   Nil = `?` .
```

Явный атрибут состоит из имени роли атрибута, последующего знака ->, последующей величины области значений роли и завершающей точки с запятой. Величина области значений роли для обязательного атрибута может быть ссылкой на экземпляр объекта или типа, значением, поименованным значением, константой или параметром, либо агрегациями перечисленных значений. Величина области значений роли для необязательного атрибута является такой же, как и для обязательного атрибута, с дополнительным значением Nil, если величина области не определена.

Примечание – Явному атрибуту может быть задано значение Nil. В этом случае если определение объекта основано на EXPRESS-объекте, тогда экземпляр не будет соответствовать EXPRESS-определению.

Пример 27 – Явные атрибуты

```
a_real          -> 1.2;
an_integer      -> 3;
a_list          -> (1, 2, 3);
a_boolean       -> TRUE;
a_logical       -> UNKNOWN;
an_enumeration  -> !enum1;
a_string        -> `A string`;
entity_ref      -> @instance2;
optional_str    -> ?;
optional_int    -> 42;
a_parameter     -> par1;
a_constant      -> c1;
```

8.7.1.2 *Вычисляемый атрибут*

Вычисляемым является атрибут, значение которого может быть вычислено по значениям других свойств объекта.

```
Синтаксис:
60i   DerivedAttr = RoleName [ `<-` DerattValue ] `;` .
107i RoleName = attribute_ref .
59i   DerattValue = AggregationValue | BaseValue | EntityInstanceRef |
                    EntityInstanceValue | EnumerationInstanceValue |
                    TypeInstanceRef | TypeInstanceValue | TypeValue .
```

Вычисляемый атрибут состоит из имени роли атрибута, последующих необязательных знака <- и величины области значений роли и заканчивается точкой с запятой. Величина области значений роли может быть ссылкой на экземпляр объекта или типа, значением, константой либо их агрегациями. Кроме того, величина может иметь значение Nil, если она не определена.

Пример 28 – Вычисляемые атрибуты

```
a_real              <- 1.2 ;
an_integer          <- 3;
a_boolean           <- TRUE;
a_logical;
an_enumeration      <- !enum1;
a_string            <- `A string`;
entity_ref          <- @instance2;
null_derived        <- ?;
```

8.7.1.3 *Инверсный атрибут*

Если экземпляр объекта установил отношение с текущим экземпляром объекта посредством ссылки в явном атрибуте на текущий экземпляр, то для описания этого отношения в контексте текущего экземпляра может быть использован инверсный атрибут.

```
Синтаксис:
87i  InverseAttr = RoleName [ `<-` InvattValue ] `;` .
107i RoleName = attribute_ref .
86i  InvattValue = DynamicEntityRefList .
62i  DynamicEntityRefList = `(` [ EntityRefList ] `)` .
71i  EntityRefList = EntityInstanceRef { `,` EntityInstanceRef } .
```

Инверсный атрибут состоит из имени роли атрибута, последующих необязательных знака <- и величины области значений и заканчивается точкой с запятой. Величиной области значений роли является (возможно, пустой) динамический список ссылок на экземпляры объектов.

Пример 29 – Инверсные атрибуты

```
inverse_1     <- (@a1, @b3);
inverse_2;
inverse_3     <- ( );
```

8.7.2 Супертипы и подтипы

Экземпляр сложного EXPRESS–объекта наследует атрибуты и их значения из экземпляров его супертипов (SUPERTYPE) (при их наличии) и передает атрибуты и их значения экземплярам своих подтипов (SUBTYPE) (при их наличии).

```
49i  BequeathesTo = SUPOF DynamicSupSubRefList `;` .
85i  InheritsFrom = SUBOF DynamicSupSubRefList `;` .
65i  DynamicSupSubRefList = `(` [ SupSubRef { `,` SupSubRef } ] `)` .
43i  SupSubRef = `@` SupSubId .
```

Ссылки на экземпляры компонентов (см. 8.7) непосредственного супертипа(ов), при его наличии, даются вслед за ключевом словом SUBOF и заключаются в круглые скобки.

Ссылки на экземпляры компонентов непосредственных подтипов, при их наличии, даются вслед за ключевым словом SUPOF и заключаются в круглые скобки.

Примечание – Как указано в 8.7, идентификатор экземпляра сложного объекта имеет две части: первая часть является идентификатором экземпляра в целом, а вторая часть – идентификатором компонента. Назовем, например, **part1** первую часть идентификатора экземпляра сложного объекта. Тогда ссылка на компонент, скажем, **@3**, является ссылкой на компонент экземпляра сложного объекта, полностью идентифицируемую как **part1[3]**.

Пример 30 – Супертипы и подтипы

```
i1[1] = super{super_int -> 2;  SUPOF(@2); };   - - имеет подтип i1[2].
i1[2] = sub{SUBOF(@1);  sub_real -> 23.7; };   - - имеет супертип i1[1].
i2[1] = sub{SUBOF(@5);  sub_real -> -42.0; }; - - имеет супертип i2[5].
i2[5] = super{super_int ->  7; SUPOF(@1); };   - - имеет подтип i2[1].
```

8.8 Экземпляр константы

Объявление константы может быть использовано для объявления именованных констант. Областью действия идентификаторов констант, объявленных внутри блока констант, должна быть схема, в которой находится блок констант. Именованная константа, появляющаяся в объявлении константы, имеет явную инициализацию, значение константы не может быть модифицировано после инициализации. Вхождение именованной константы вне ее объявления должно быть эквивалентно вхождению исходного значения самой константы.

Синтаксис:

```
52i ConstantBlock = CONSTANT { ConstantSpec } END_CONSTANT `;` .
54i ConstantSpec = ConstantId `= =` ConstantValue `;` .
53i ConstantValue = AggregationValue | BaseValue | EntityInstanceValue |
                    NamedInstanceValue | SelectValue | TypeValue .
35i ConstantRef = ConstantId .
```

Значение константы может быть агрегацией значений.

Правила и ограничения

a) Каждое значение должно быть простым значением, значением экземпляра объекта, перечисляемым значением, выбираемым значением либо их агрегациями.

b) Именованная константа может появляться в объявляемом значении другой именованной константы.

Пример 31 – Блок констант

```
CONSTANT
    zero          = = 0.0;
    thousand      = = 1000;
    origin        = = point{x -> zero; y -> zero;};
    large_circle  = = circle{center -> origin; radius -> thousand;};
    z_axis        = = [0.0, 0.0, 1.0];
END_CONSTANT;
```

8.9 Экземпляр данных схемы

Экземпляр данных схемы (SCHEMA_DATA) определяет экземпляр (часть) представления рассматриваемой предметной области, в котором объявляемые элементы имеют определенные смысл и назначение. Например, геометрия (**geometry**) может быть именем данных схемы (SCHEMA_DATA), содержащих экземпляры точек, кривых, поверхностей и других соответствующих элементов. Порядок, в котором объявляются экземпляры в экземпляре SCHEMA_DATA, произволен.

Синтаксис:

```
109i SchemaInstanceBlock = SCHEMA_DATA SchemaId `;`
                           [ SchemaInstanceBody ] END_SCHEMA_DATA `;` .
108i SchemaId = schema_ref .
110i SchemaInstanceBody = [ ConstantBlock ] { ObjectInstance } .
95i  ObjectInstance = EntityInstance | EnumerationInstance | SelectInstance |
                      TypeInstance | SimpleInstance .
```

Объявление SCHEMA_DATA создает новую область действия, в которой могут быть объявлены следующие элементы:

- константы;
- экземпляры объектов;
- экземпляры перечисления;
- экземпляры выбора;
- простые экземпляры;
- экземпляры типа.

Пример 32 – Наполнение EXPRESS-схемы

```
SCHEMA_DATA  whatsits ;
    (* Константы, определяемые в EXPRESS *)
    CONSTANT
```

```
one  =  1.0;
twopi  =  6.2831853;
END_CONSTANT;
(* Типы, определяемые в EXPRESS *)
n1  =  name{(`Jot`, `E`, `Bloggs`)};
n2  =  name{(`Mary`, `Jones`)};
(* Объекты, определяемые в EXPRESS *)
p1  =  point{x -> one; y -> twopi;};
s1  =  affianced{him -> @n1; her -> @n2;};
END_SCHEMA_DATA;
```

8.10 Отображение модели

Понятие MODEL определяет одну конкретную реализацию данных, соответствующих информационной модели.

```
Синтаксис:
90i  ModelBlock  =  MODEL  ModelId  `;`  ModelBody  END_MODEL  `;`  .
92i  ModelId  =  simple_id  .
91i  ModelBody  =  {  SchemaInstanceBlock  }  .
```

EXPRESS-объявление MODEL создает новую область действия, в которой могут быть объявлены следующие элементы:

- экземпляры данных схемы.

Примечание – Основным назначением MODEL является демонстрация совокупности объектной базы.

Пример 33 – Например, **bugatti_35** может быть именем MODEL, содержащей данные, представляющие автомобиль типа *Bugatti Type 35.* Может быть несколько экземпляров данных схемы внутри MODEL: один, например, для чертежей автомобиля, и другой, содержащий данные по обслуживанию автомобиля этого типа.

Правила и ограничения

a) Каждый экземпляр данных схемы внутри MODEL должен иметь уникальный идентификатор.

b) Идентификатор каждого экземпляра внутри MODEL должен быть уникальным.

c) Значения внутри MODEL не должны быть ссылками на параметр.

Пример 34 – Набросок MODEL

```
MODEL  a_model;
   SCHEMA_DATA  a_schema;
   ...
   END_SCHEMA_DATA;
   SCHEMA_DATA  another_schema;
   ...
   END_SCHEMA_DATA;
END  MODEL.
```

9 Определение абстрактного контрольного примера

В данном разделе описаны основные элементы языка EXPRESS-I, относящиеся к определению абстрактных контрольных примеров.

9.1 Контекст

Контекст (CONTEXT) определяет экземпляры данных и алгоритмы, относящиеся к представлению рассматриваемой предметной области, в котором элементы имеют определенные смысл и назначение. Экземпляры данных могут быть параметризованы.

Синтаксис:
56i ContextBlock = CONTEXT ContextId `;` ContextBody END_CONTEXT `;` .
58i ContextId = simple_id .
57i ContextBody = { SchemaReferenceSpec } [FormalParameterBlock]
{ SchemaInstanceBlock | SupportAlgorithm } .
36i ContextRef = ContextId .

EXPRESS-I-объявление CONTEXT создает новую область действия, в которой могут быть объявлены следующие элементы:

- ссылки на EXPRESS-схемы (см. 10.2);
- формальные параметры;
- экземпляры данных схемы;
- EXPRESS-функции;
- EXPRESS-процедуры.

Пример 35 – Например **bugatti** может быть именем CONTEXT, который содержит параметризованные (то есть обобщенные) данные, представляющие автомобиль типа *Bugatti.* Внутри этого CONTEXT может быть несколько экземпляров данных схемы: один, например, для чертежей автомобиля, и другой, содержащий данные по обслуживанию автомобиля этого типа.

Правила и ограничения

a) Каждый экземпляр данных схемы внутри CONTEXT должен быть экземпляром из разных схем.

b) Каждый идентификатор внутри CONTEXT должен быть уникальным.

Пример 36 – Набросок CONTEXT

```
CONTEXT parametrized_model;
   PARAMETER
   ...
   END_PARAMETER;
   SCHEMA_DATA  a_schema;
   ...
   END_SCHEMA_DATA;
   SCHEMA_DATA another_schema;
   ...
   END_SCHEMA_DATA;
END_CONTEXT;
```

9.2 Параметры

Контекст может иметь формальные параметры. Каждый формальный параметр имеет имя и область значений. Имя является идентификатором, который должен быть уникальным внутри области действия контекста.

Контрольный пример может иметь фактические параметры, которые задают конкретные значения соответствующим формальным параметрам внутри контекста.

Для обобщения типов данных, используемых для передачи значений в контекстах, имеются области значений AGGREGATE и GENERIC. Также могут использоваться соответствующие массивы для обобщения областей значений массивов.

9.2.1 Формальный параметр

Формальный параметр может иметь значение по умолчанию, которое должно быть совместимо с областью значений. Формальные параметры, не имеющие значений по умолчанию, изначально обнуляются (имеют значение) Nil.

```
Синтаксис:
83i  FormalParameterBlock = PARAMETERi
                            { FormalParameter } END_PARAMETER `;` .
82i  FormalParameter = ParameterId `:` parameter_type
                       [ `:=` ParmValueDefault ] `;` .
100i ParameterId = simple_id .
253  parameter_type = < как в EXPRESS > .
103i ParmValueDefault = AggregationValue | BaseValue | ConstantRef |
                        EntityInstanceValue | NamedInstanceValue |
                        ObjectInstanceRef | SelectValue | TypeValue |
                        expression .
204  expression = < как в EXPRESS > .
39i  ParameterRef = ParameterId .
```

Так как в контексте могут быть несколько экземпляров данных схем, содержащих параметры, может случиться, что две или более схемы имеют объекты или типы с одинаковыми именами, но разной семантикой. Использование одного из этих имен в качестве идентификатора области значений параметра оказалось бы в этом случае двусмысленным. В случаях потенциальной неоднозначности каждое имя должно быть квалифицировано предшествующим ему именем соответствующей схемы с точкой в качестве разделителя.

Пример 37 – Блок PARAMETER

```
PARAMETER
    iv1          : INTEGER := 1;
    bv1          : BOOLEAN;
    p1           : name := name{first -> 'John'; last -> 'Doe'; married -> bv1;};
    p2           : name := name(`Mary`, `Smith`, TRUE);
    a_list       : LIST OF REAL := (0.0, 1.0, 2.0);
    a_set        : SET OF STRING;
    a_select     : selection := wheeled_vehicle;
    from_sch1    : sch1.vector := [1.0, 3.0];
    from_sch2    : sch2.vector := [3.0, 4.0, -0.5];
END_PARAMETER;
```

9.2.2 Фактический параметр

Фактический параметр состоит из ссылки на формальный параметр и значения параметра. Значение должно быть совместимым с областью значений формального параметра. Значение заменяет значение параметра по умолчанию, заданное формальным параметром.

```
Синтаксис:
45i  ActualParameter = ParameterRef `:=` ParmValue .
39i  ParameterRef = ParameterId .
102i ParmValue = ObjectInstanceRef | expression .
204  expression = < как в EXPRESS > .
```

Пример 38 — Пример показывает некоторые фактические параметры для формальных параметров, заданных в примере 37.

```
iv1          := 77**2;
bv1          := FALSE;
p1           := name(`John`, `Smith`, bv1);
a_list       := [20.0, 1.0, 20.0, 33.72];
a_select     := @v23;
from_sch1    := [0.0, -1.0];
from_sch2    := [0.5, -0.2, -0.15];
```

9.3 Контрольный пример

Тип TEST_CASE определяет как управляющие данные, так и данные экземпляра, которые могут быть использованы для целей абстрактного контрольного примера.

```
Синтаксис:
127i TestCaseBlock = TEST_CASE TestCaseId `;`
                         TestCaseBody END_TEST_CASE `;` .
129i TestCaseId = simple_id .
128i TestCaseBody = SchemaReferences ObjectiveBlock TestRealization
                        { SupportAlgorithm } .
111i SchemaReferences = SchemaReferenceSpec { SchemaReferenceSpec } .
```

Объявление TEST_CASE создает новую область действия, в которой можно объявлять либо ссылаться на следующие элементы:

- тестируемые элементы (см. 10.2);
- цель теста;
- реализацию теста;
- алгоритмы поддержки.

Тип TEST_CASE позволяет ссылаться на одну или несколько EXPRESS-схем. Это могут быть ссылки на набор контекстов (CONTEXT) и, возможно, на набор значений параметров, предназначенных для определения набора тестовых данных.

Правила и ограничения

a) Значение каждого фактического параметра, объявленного в контрольном примере, должно быть совместимо с областью значений соответствующего формального параметра, объявленного в контексте.

b) Значение контрольного примера, связанное с каждым формальным параметром из контекста, должно быть объявлено в качестве фактического параметра или принято как значение формального параметра по умолчанию, если фактический параметр не объявляется.

c) Типы данных внутри контрольного примера должны ограничиваться определениями типов, установленными внутри ссылочных схем.

9.4 Цель теста

Целью теста (OBJECTIVE) являются управляющие данные, которые могут быть использованы для абстрактного контрольного примера.

```
Синтаксис:
97i ObjectiveBlock = OBJECTIVE { TestPurpose } { TestReference }
                         { TestCriteria } { TestNotes }
                     END_OBJECTIVE `;` .
```

Объявление OBJECTIVE создает новую область действия, в которой могут быть объявлены следующие элементы:

- назначение абстрактного контрольного примера;
- ссылки на соответствующие стандарты или технические требования;
- критерий теста;
- замечания для аналитика теста.

Пример 39 – Цель теста

```
OBJECTIVE
    NOTES Настоящая цель содержит только замечания для аналитика теста.
    END_NOTES;
END_OBJECTIVE;
```

9.4.1 Назначение теста

Назначением теста является текст, адресуемый человеку. Он содержит описание целевого назначения теста.

```
Синтаксис:
133i TestPurpose = PURPOSE Description END_PURPOSE `;` .
26i  Description = { \a | \s | \n } .
```

Текст начинается с ключевого слова PURPOSE и заканчивается ключевым словом END_PURPOSE и точкой с запятой. Текст может занимать несколько строк.

Пример 40 – Текст в данном случае занимает две строки.

PURPOSE. This test is intended to check the existance of a car instance. (Настоящий тест предназначен для проверки наличия экземпляра автомобиля). END_PURPOSE;

9.4.2 Тестовые ссылки

Тестовой ссылкой является текст, адресуемый человеку. Он содержит описание интерпретируемых человеком ссылок на соответствующие стандарты или технические требования (спецификации).

```
Синтаксис:
134i TestReference = REFERENCES Description END_REFERENCES `;` .
26i  Description = { \a | \s | \n } .
```

Текст начинается с ключевого слова REFERENCES и заканчивается ключевым словом END_REFERENCES и точкой с запятой. Текст может занимать несколько строк.

Пример 41 – Ссылка на печатный документ.

REFERENCES Документ AP279, страницы 53—57. END_REFERENCES;

9.4.3 Критерий теста

Критерием теста является текст, адресуемый человеку. Он содержит описание критерия вердикта, используемого при оценке результатов теста.

```
Синтаксис:
131i TestCriteria = CRITERIA Description END_CRITERIA `;` .
26i  Description = { \a | \s | \n } .
```

Текст начинается с ключевого слова CRITERIA и заканчивается ключевым словом END_CRITERIA и точкой с запятой. Текст может занимать несколько строк.

Пример 42 – Простой критерий

CRITERIA At least one instance of car shall be present. (Должен быть представлен по крайней мере один экземпляр автомобиля). END_CRITERIA;

9.4.4 Замечания к тесту

Замечаниями к тесту является текст, адресуемый человеку. Он обеспечивает способы описания общих замечаний, помогающих аналитику теста.

```
Синтаксис:
132i TestNotes = NOTES Description END_NOTES `;` .
26i  Description = { \a | \s | \n } .
```

Текст начинается с ключевого слова NOTES и заканчивается ключевым словом END_NOTES и точкой с запятой. Текст может занимать несколько строк.

Пример 43 – Замечание в одну строку

NOTES Remember to fasten your seat belt. (Не забудьте закрепить ремень безопасности). END_NOTES;

9.5 Реализация теста

Реализация теста дает определение элементов данных, относящихся к контрольному примеру.

```
Синтаксис:
130i TestRealization = REALIZATION { local_decl } { UseContextBlock }
                       { assignment_stmt } END_REALIZATION `;` .
239  local_decl = < как в EXPRESS > .
166  assignment_stmt = < как в EXPRESS > .
```

Реализация начинается с ключевого слова REALIZATION и заканчивается ключевым словом END_REALIZATION и точкой с запятой.

Реализация теста может включать:

- ссылки на данные контекста и параметры (см. 10.3.);
- локальные переменные (определяемые с помощью синтаксиса EXPRESS);
- операторы присваивания (определяемые с помощью синтаксиса EXPRESS).

Пример 44 – Данная реализация определяет **p1** как переменную типа **point** (точка). Данный тип затем вызывается для создания точки с координатами (1, 2, 3), присваиваемыми экземпляру переменной **p1**.

```
REALIZATION
   LOCAL
      p1 : point;
   END_LOCAL;

   p1 : point(1.0, 2.0, 3.0);
END_REALIZATION;
```

10 Интерфейсы

В данном разделе установлены интерфейсы между экземплярами EXPRESS-I и EXPRESS-моделями вместе с интерфейсами между конструкциями EXPRESS-I.

10.1 Интерфейс экземпляра схемы

```
Синтаксис:
109i  SchemaInstanceBlock = SCHEMA_DATA SchemaId;
                            [ SchemaInstanceBody ] END_SCHEMA_DATA `;` .
108i  SchemaId = schema_ref .
152   schema_ref = < как в EXPRESS > .
```

Предполагается, что имеется связанная EXPRESS-схема (или, что то же самое, EXPRESS-G-схема); тогда SchemaId ссылается на имя этой EXPRESS-схемы. Это значит, что тело экземпляра данных EXPRESS-I-схемы содержит экземпляры данных определений из идентифицированной EXPRESS-схемы. Оно не должно содержать экземпляров данных определений, являющихся внешними по отношению к данной EXPRESS-схеме.

Примечание – Ссылки на схемы, определяемые в языках, отличных от EXPRESS или EXPRESS-G, не являются предметом рассмотрения в настоящем стандарте. Тем не менее SchemaId можно рассматривать как ссылку на схему, определяемую на языке, отличном от EXPRESS.

10.2 Ссылка на схему

Ссылка на схему позволяет идентифицировать конкретную EXPRESS-схему вместе с конкретными определениями внутри этой схемы.

```
Синтаксис:
112i  SchemaReferenceSpec = WITH schema_ref [ USING `(` resource_ref
                            { `,` resource_ref } `)` ] `;` .
152   schema_ref = < как в EXPRESS > .
275   resource_ref = < как в EXPRESS > .
```

Конструкция **schema_ref**, следующая за ключевым словом WITH, обозначает конкретную EXPRESS-схему. Конкретные объявления, представляющие интерес в данной EXPRESS-схеме, обозначаются в списке, следующем за ключевым словом USING.

Отсутствие списка USING означает, что все определения внутри обозначенной EXPRESS-схемы являются доступными.

Примечание – Ссылка на схему действует аналогично EXPRESS-оператору USE.

Пример 45 – Задано следующее EXPRESS-определение:

```
SCHEMA a_schema;
      ENTITY entity1; ... END_ENTITY;
      ENTITY entity2; ... END_ENTITY;
      ENTITY entity7; ... END_ENTITY;
```

```
    TYPE  type19 = ...  END_TYPE;
    TYPE  type21 = ...  END_TYPE;
END_SCHEMA;
```

Тогда следующая конструкция обозначает два объекта и один тип из схемы **a_schema**.

```
WITH  a_schema  USING  (entity1,  entity7,  type21);
```

10.3 Ссылки на данные контекста

Элементы из CONTEXT могут быть импортированы в TEST_CASE, а фактические значения могут быть заданы для формальных параметров в CONTEXT.

```
Синтаксис:
141i  UseContextBlock  =  CALL  ContextRef  `;`  UseContextBody  END_CALL  `;`  .
36i   ContextRef  =  ContextId  .
142i  UseContextBody  =  [  ImportSpec  ]  [  ParameterSpec  ]  .
84i   ImportSpec  =  IMPORT  `(`  {  Assignment  }  `)`  `;`  .
47i   Assignment  =  variable_id  `:=`  SelectableInstanceRef  `;`  .
101i  ParameterSpec  =  WITH  `(`  {  ActualParameter  }  `)`  `;`  .
113i  SelectableInstanceRef  =  EntityInstanceRef  |  EnumerationInstanceRef  |
                                SelectInstanceRef  |  TypeInstanceRef  .
```

Конкретный CONTEXT обозначается посредством оператора CALL.

Экземпляры предметов, представляющие интерес для контрольного примера и существующие в CONTEXT, обозначаются в списке IMPORT. Каждое значение экземпляра должно быть присвоено переменной.

Значения формальных параметров в CONTEXT (при их наличии) задаются посредством списка WITH. Эти значения должны отменять значения обозначенных параметров, принятые по умолчанию (при их наличии).

Пример 46 – Спецификация CALL

```
CALL  a_context;
   IMPORT (ent_var  :=  @ent_21;
                ent_27  :=  @ent_27;);
   WITH  (iv1  :=  771;
           a_set  :=  [  `alpha`,  `to`,  `omega`];  );
END_CALL;
```

11 Область действия и видимость

Объявление EXPRESS-I создает идентификатор, который можно использовать для ссылки на объявленный элемент в других контекстах. Некоторые конструкции EXPRESS-I неявно объявляют элементы EXPRESS путем добавления к ним идентификаторов. В тех областях, где на идентификатор объявленного элемента можно ссылаться, объявленный элемент называется видимым. На элемент можно ссылаться только там, где идентификатор видим. Относительно правил видимости см. 11.2.

Некоторые элементы EXPRESS-I определяют участок (блок) текста, называемый областью действия элемента. Эта область действия ограничивает видимость объявленных в ней идентификаторов. Области действия могут быть вложенными; это значит, что элемент EXPRESS-I, устанавливающий область действия, может быть включен в область действия другого элемента. Имеются ограничения на перечень элементов, которые могут появиться внутри области действия конкретного элемента EXPRESS-I. Эти ограничения обычно устанавливаются синтаксисом EXPRESS-I (см. приложение А).

Для каждого из элементов, установленных в таблицах 9 и 10, последующие подразделы настоящего раздела устанавливают границы определяемой области действия (при их наличии) и видимость объявляемого идентификатора как в общих терминах, так и в конкретных деталях.

Таблица 9 – Область действия и идентификатор, определяющие элементы EXPRESS-I

Элемент	Область действия	Идентификатор
экземпляр константы (constant instance)		•
контекст (context)	•	•
экземпляр объекта (entity instance)		•
экземпляр перечисления (enumeration instance)		•
модель (model)	•	•
экземпляр данных схемы (schema data instance)	•	•
экземпляр выбора (select instance)		•
простой экземпляр (simple instance)		•
контрольный пример (test case)	•	•
экземпляр типа (type instance)		•

Примечание – EXPRESS-I использует также различные конструкции EXPRESS, которые аналогичным образом имеют идентификаторы и области действия. Они приведены в таблице 10.

11.1 Правила области действия

Ниже приведены общие правила, применяемые для всех форм определения области действия, допустимых в языке EXPRESS-I; список элементов, для которых определяются области действия, – см. в таблицах 9 и 10.

Правила и ограничения

a) Все объявления должны находиться внутри области действия.

b) Внутри одной области действия идентификатор можно объявить только один раз либо явно его импортировать из вне.

c) Области действия должны быть корректно вложенными, то есть не должны пересекаться (это предписывается синтаксисом языка).

Таблица 10 – Области действия и идентификаторы, определяющие элементы EXPRESS и используемые в EXPRESS-I

Элемент	Область действия	Идентификатор
оператор переименования (alias statement)	•	• [1]
атрибут (attribute)		•
константа (constant)		•
объект (entity)	•	•
перечисление (enumeration)		•
функция (function)	•	•
параметр (parameter)		•
процедура (procedure)	•	•
выражение запроса (query expression)	•	• [1]
оператор цикла (repeat statement)	•	• [1, 2]
метка правила (rule label)		•
тип (type)	•	•
метка типа (type label)		•
переменная (variable)		•

Примечания

1 Идентификатор является неявно объявляемой переменной внутри определяемой области действия объявления.

2 Переменная неявно объявляется только тогда, когда устанавливается управление циклом по приращению.

В настоящем стандарте не устанавливается максимально допустимая глубина вложенности. Разработчик синтаксического анализатора языка EXPRESS-I должен установить максимальную глубину вложенности, поддерживаемую данной реализацией (см. приложение В).

11.2 Правила видимости

Правила видимости идентификаторов описаны ниже. Список элементов EXPRESS-I, для которых объявляются идентификаторы, см. в таблицах 9 и 10. Правила видимости идентификаторов поименованных типов данных несколько отличаются от правил видимости других идентификаторов. Эти отличия описаны в 11.2.2.

11.2.1 Общие правила видимости

Приведенные ниже общие правила применимы ко всем идентификаторам, за исключением идентификаторов поименованных типов данных, на которые правило d) не распространяется.

Правила и ограничения

a) Идентификатор видим в области действия, в которой он объявляется. Эта область действия называется локальной областью действия идентификатора.

b) Идентификатор видим в конкретной области действия, он также видим во всех областях действия, определяемых внутри данной области, согласно правилу d).

c) Идентификатор не видим ни в одной области действия за пределами его локальной области действия, согласно правилу f).

d) Если идентификатор *i*, видимый в области действия *P*, переобъявляется в некоторой внутренней области *Q*, заключенной в *P*, только *i*, объявленный в области действия *Q*, видим в *Q* и областях действия, объявленных внутри *Q*. Идентификатор *i*, объявленный в области действия *P*, видим в *P* и любых внутренних областях действия, не переобъявляющих *i*.

e) Встроенные константы, функции, процедуры и типы EXPRESS-I считаются объявленными в воображаемой универсальной области действия. Все области действия EXPRESS-I являются вложенными в эту область действия. Идентификаторы, по которым ссылаются на встроенные константы, функции, процедуры и типы EXPRESS-I, видимы во всех областях действия, определяемых EXPRESS-I.

f) Идентификаторы элементов перечисления, объявленные внутри области определенного типа данных, видимы в следующей внешней области действия, если она не содержит объявления этого же идентификатора для другого элемента.

Примечание – Если следующая внешняя область действия содержит объявление того же идентификатора, элементы перечисления остаются доступными, но должны иметь префиксом идентификатор определенного типа данных.

g) Некоторые EXPRESS-I-объявления, обычно не видимые, могут быть сделаны видимыми с помощью спецификаций интерфейса (см. раздел 10).

11.2.2 Правила видимости идентификатора поименованного типа данных

За одним исключением, идентификаторы поименованных типов данных подчиняются тем же правилам видимости, что и другие идентификаторы. Этим исключением является правило видимости d). Идентификатор объекта или определенного типа данных *i*, объявленный в области действия *P*, остается видимым во внутренней области *Q*, даже если он переопределяется в *Q* одним из следующих способов:

a) область действия *Q* определяется объявлением объекта, а *i* объявляется как атрибут в этой области действия;

b) область действия *Q* определяется объявлением функции, процедуры или контекста, а *i* объявляется как формальный параметр или переменная в этой области действия.

Пример 47 – В **entity1 d** относится как к типу данных объекта, так и к атрибуту.

```
FUNCTION example (par : INTEGER): INTEGER;
  ENTITY d;
    attr1 : REAL;
  END_ENTITY;
  ENTITY entity1;
    d : d; - - d в данной области действия является и объектом и атрибутом.
  END_ENTITY;
  ...
END_FUNCTION;
```

11.3 Правила для явного элемента

В последующих пунктах настоящего подраздела более детально показано, как общие правила области действия и видимости применяются к различным элементам EXPRESS-I.

В EXPRESS-I используется многое из языка EXPRESS. Правила области действия и видимости для большинства этих элементов EXPRESS внутри EXPRESS-I идентичны правилам EXPRESS, определенным в ГОСТ Р ИСО 10303-11. В таблице 11 определены эти элементы. Кроме того, в таблице 11 определены элементы, общие для EXPRESS и EXPRESS-I, для которых EX-

PRESS-правила модифицируются при их использовании в EXPRESS-I, а также элементы, специфические для EXPRESS-I.

Таблица 11 – Правила области действия и видимости

Элемент	Правила EXPRESS	Модифицированные правила EXPRESS	Специфические правила EXPRESS-I
оператор переименования	•		
атрибут	•		
константа		•	
экземпляр константы			•
контекст			•
объект		•	
экземпляр объекта			•
перечисление		•	
экземпляр перечисления			•
функция		•	
модель			•
параметр		•	
процедура		•	
выражение запроса	•		
оператор цикла	•		
метка правила		•	
экземпляр данных схемы			•
экземпляр выбора			•
простой экземпляр			•
контрольный пример			•
тип		•	
экземпляр типа			•
метка типа	•		
переменная		•	

Примечание – Модификации EXPRESS-правил обусловлены в основном, тем, что EXPRESS-I не использует EXPRESS-конструкций SCHEMA или RULE.

11.3.1 Оператор переименования

Правила области действия и видимости для оператора ALIAS определены в 10.3.1 ГОСТ Р ИСО 10303-11.

11.3.2 Атрибут

Правила области действия и видимости для атрибута определены в 10.3.2 ГОСТ Р ИСО 10303-11.

11.3.3 Константа

Видимость: Идентификатор константы видим в области действия функции или процедуры, в которой он объявлен.

Примечание – EXPRESS-спецификация (10.3.3 ГОСТ Р ИСО 10303-11) гласит: идентификатор константы видим в области действия функции, процедуры, правила или схемы, в которых он объявлен.

11.3.4 Экземпляр константы

Видимость: Идентификатор экземпляра константы видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия экземпляра данных схемы.

11.3.5 Контекст

Видимость: Идентификатор контекста видим для всех контрольных примеров.

Область действия: Объявление контекста определяет новую область действия. Эта область действия начинается с ключевого слова CONTEXT и продолжается до ключевого слова END_CONTEXT, которым заканчивается объявление контекста.

Объявления: Следующие элементы могут объявлять идентификаторы в области действия объявления контекста:

- формальный параметр;
- функция;
- процедура;
- экземпляр данных схемы.

11.3.6 Объект

Видимость: Идентификатор объекта видим в области действия функции или процедуры, в которой он объявлен. Идентификатор объекта остается видимым при условиях, определенных в 11.2.2, во внутренних областях действия, которые переобъявляют этот идентификатор.

Примечание – EXPRESS-спецификация (10.3.5 ГОСТ Р ИСО 10303-11) гласит: идентификатор объекта видим в области действия функции, процедуры, правила или схемы, в которых он объявлен. Идентификатор объекта остается видимым ....

Область действия и объявления: Область действия и допустимые объявления определены в ГОСТ Р ИСО 10303-11.

Пример 48 – Идентификаторы атрибута **batt** в двух объектах не взаимосвязаны, поскольку они объявлены в двух разных областях действия.

```
ENTITY entity1;
   aatt : INTEGER;
   batt : INTEGER;
END_ENTITY;

ENTITY entity2;
   a          : entity1;
   batt: INTEGER;
END_ENTITY;
```

Пример 49 – Следующая спецификация является неправильной, так как идентификатор атрибута **aatt** повторяется внутри области действия одного объекта. Хотя метка правила **lab** объявлена в обоих объектах, это не нарушает правил области действия или видимости; объявление объекта **may_be_ok** не видимо в объекте **illegal**, но правила обоих областей значений должны быть проверены.

```
ENTITY may_be_ok;
   quantity : REAL;
WHERE
   lab : quantity >= 0.0;
END_ENTITY;

ENTITY illegal
SUBTYPE OF (may_be_ok);
   aatt : INTEGER;
   batt : INTEGER;
   aatt : REAL;
WHERE
   lab : batt < 0;
END_ENTITY;
```

11.3.7 Экземпляр объекта

Видимость: Идентификатор экземпляра объекта видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия этого экземпляра данных схемы.

11.3.8 Элемент перечисления

Видимость: Идентификатор элемента перечисления видим в области действия функции или процедуры, в которой объявлен его тип. Это является исключением из правила видимости 11.2.1f. Идентификатор не должен объявляться с какой-либо иной целью в этой области действия, за исключением объявления другого перечисляемого типа данных в той же области действия. Если один и тот же идентификатор объявляется двумя перечисляемыми типами данных как элемент перечисления, к ссылке на каждый элемент перечисления должен быть добавлен префикс в виде идентификатора типа данных для обеспечения однозначности ссылки.

Примечание – EXPRESS-спецификация (10.3.4 ГОСТ Р ИСО 10303-11) гласит: идентификатор элемента перечисления видим в области действия функции, процедуры, правила или схемы, в которой объявлен его тип. Это является исключением ....

11.3.9 Экземпляр перечисления

Видимость: Идентификатор экземпляра перечисления видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия этого экземпляра данных схемы.

11.3.10 Функция

Видимость: Идентификатор функции видим в области действия функции, процедуры, контекста или контрольного примера, в которых он объявлен.

Примечание – EXPRESS-спецификация (10.3.6 ГОСТ Р ИСО 10303-11) гласит: идентификатор функции видим в области действия функции, процедуры, правила или схемы, в которых он объявлен.

Область видимости и объявления: Область действия и допустимые объявления определены в ГОСТ Р ИСО 10303-11.

11.3.11 Модель

Область действия: Объявление модели определяет новую область действия. Эта область действия простирается от ключевого слова MODEL до ключевого слова END_MODEL, которым заканчивается объявление модели.

Объявления: Следующие элементы могут объявлять идентификаторы в области действия объявления модели:

- экземпляр данных схемы.

11.3.12 Параметр

Видимость: Идентификатор формального параметра видим в области действия функции, процедуры или контекста, в которых он объявлен.

Примечание – EXPRESS-спецификация (10.3.7 ГОСТ Р ИСО 10303-11) гласит: идентификатор формального параметра видим в области действия функции или процедуры, в которых он объявлен.

Пример 50 – Следующее объявление является неправильным, так как идентификатор формального параметра **parm** также используется в качестве идентификатора локальной *переменной*:

```
CONTEXT illegal;
   PARAMETER
      parm : REAL;
      ...
   END_PARAMETER;
   LOCAL
      parm : STRING;
   END_LOCAL;
   ...
END_CONTEXT;
```

11.3.13 Процедура

Видимость: Идентификатор процедуры видим в области действия функции, процедуры, контекста или контрольного примера, в которых он объявлен.

Примечание – EXPRESS-спецификация (10.3.8 ГОСТ Р ИСО 10303-11) гласит: идентификатор процедуры видим в области действия функции, процедуры, правила или схемы, в которых он объявлен.

Область действия и объявления: Область действия и допустимые объявления определены в 10.3.8 ГОСТ Р ИСО 10303-11.

11.3.14 Выражение запроса

Область действия и видимость выражения QUERY определены в 10.3.9 ГОСТ Р ИСО 10303-11.

11.3.15 Оператор цикла

Область действия и видимость оператора REPEAT определены в 10.3.10 ГОСТ Р ИСО 10303–11.

11.3.16 Метка правила

Видимость: Метка правила видима в области действия объекта или типа, в которых она объявлена.

Примечания

1 EXPRESS-спецификация (10.3.12 ГОСТ Р ИСО 10303-11) гласит: метка правила видима в области действия объекта, правила или типа, в котором она объявлена.

2 Метка правила используется только в реализации. EXPRESS-I не определяет механизма для ссылок на метки правила.

11.3.17 Экземпляр данных схемы

Область действия: Объявление данных схемы определяет новую область действия. Эта область простирается от ключевого слова SCHEMA_DATA до ключевого слова END_SCHEMA_DATA, которым заканчивается объявление данных схемы.

Объявления: Следующие элементы могут объявлять идентификаторы в области действия объявления данных схемы:

- экземпляр константы;
- экземпляр объекта;
- экземпляр перечисления;
- экземпляр выбора;
- простой экземпляр;
- экземпляр типа.

11.3.18 Экземпляр выбора

Видимость: Идентификатор экземпляра выбора видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия этого экземпляра данных схемы.

11.3.19 Простой экземпляр

Видимость: Идентификатор простого экземпляра видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия этого экземпляра данных схемы.

11.3.20 Контрольный пример

Область действия: Контрольный пример определяет новую область действия. Эта область действия простирается от ключевого слова TEST_CASE до ключевого слова END_TEST_CASE, которым заканчивается данный контрольный пример.

Объявления: Следующие элементы могут объявлять идентификаторы в области действия контрольного примера:

- функция;
- процедура;
- переменная.

11.3.21 Тип

Видимость: Идентификатор типа видим в области действия функции или процедуры, в которой он объявлен. Идентификатор типа остается видимым, при определенных условиях, во внутренних областях действия, переобъявляющих этот идентификатор; для определения допустимых условий — см. 11.2.2.

Примечание — EXPRESS-спецификация (10.3.14 ГОСТ Р ИСО 10303-11) гласит: идентификатор типа видим в области действия функции, процедуры, правила или схемы, в которых он объявлен. Идентификатор типа остается видимым ....

Область действия и объявления: Область действия и допустимые объявления определены в ГОСТ Р ИСО 10303-11.

11.3.22 Экземпляр типа

Видимость: Идентификатор экземпляр типа видим в области действия экземпляра данных схемы, в котором он объявлен, и в любой внешней области действия этого экземпляра данных схемы.

11.3.23 Метка типа

Область действия и видимость определены в 10.3.15 ГОСТ Р ИСО 10303-11.

11.3.24 Переменная

Видимость: Идентификатор переменной видим в области действия функции, процедуры или контрольного примера, в которых он объявлен.

Примечание — EXPRESS-спецификация (10.3.16 ГОСТ Р ИСО 10303-11) гласит: идентификатор переменной видим в области действия функции, процедуры или правила, в которых он объявлен.

12 Отображение из EXPRESS в EXPRESS-I

В настоящем разделе установлены правила отображения определений схемы и типа из EXPRESS в экземпляры EXPRESS-I.

В таблице 12 приведен обзор отображений из EXPRESS в EXPRESS-I. Более детально они описаны ниже.

Таблица 12 – Краткий обзор отображений из EXPRESS в EXPRESS-I

EXPRESS	EXPRESS-I
ARRAY, BAG, LIST, SET	AggregationValue
CONSTANT	ConstantBlock ContextBlock
ENTITY	EntityInstance
ENUMERATION	Экземпляр или значение перечисления FormalParameterBlock
FUNCTION	ModelBlock
PROCEDURE	
Remark	
RULE	
SCHEMA	SchemaInstanceBlock
SELECT	Экземпляр или значение выбора
Simple type	SimpleValue TestCaseBlock
TYPE	Экземпляр или значение типа

12.1 Отображение EXPRESS-схемы

EXPRESS-конструкция SCHEMA синтаксически отображается в EXPRESS-I-конструкцию экземпляра данных схемы. В таблице 13 приведен обзор соответствия между конструкциями EXPRESS и EXPRESS-I.

Правила и ограничения

a) Имя экземпляра данных EXPRESS-I-схемы должно быть таким же, как и имя соответствующей EXPRESS-схемы.

b) Каждый экземпляр объекта внутри экземпляра данных схемы должен иметь соответствующее определение объекта внутри EXPRESS-схемы.

c) Каждый экземпляр перечисления, выбора или типа внутри экземпляра данных схемы должен иметь соответствующее определение внутри EXPRESS-схемы.

d) Каждая константа внутри экземпляра данных схемы должна иметь соответствующее определение константы внутри EXPRESS-схемы.

e) Каждая спецификация области значений внутри экземпляра данных схемы должна быть уникально обозначена, при необходимости – посредством квалификации имени области значений, именем EXPRESS-схемы, содержащей определение области значений.

f) Идентификаторы экземпляров должны быть уникальными внутри экземпляра данных схемы.

Таблица 13 – Обзор отображения SCHEMA

EXPRESS	EXPRESS-I
имя SCHEMA	schema_id
CONSTANT	ConstantBlock или ничего
ENTITY	EntityInstance
ENUMERATION	EnumerationInstance или ничего
FUNCTION	ничего
PROCEDURE	ничего
REFERENCE	ничего, но см. 12.1.1
RULE	ничего
SELECT	SelectInstance или ничего
TYPE	TypeInstance или ничего
USE	ничего, но см. 12.1.1

12.1.1 Отображение USE и REFERENCE

EXPRESS-операторы USE и REFERENCE не отображаются в EXPRESS-I непосредственно, но их действие приводит к следующему:

- экземпляры элементов EXPRESS, внесенные в область действия EXPRESS-схемы посредством явных операторов USE или REFERENCE либо посредством неявных ссылок, могут появиться внутри соответствующего экземпляра данных EXPRESS-I-схемы;

- элементы, области значений которых переименовываются, должны иметь соответствующие области значений с новыми именами;

- если имеются конфликты между именами областей значений из исходной EXPRESS-схемы с именами областей значений, вносимыми из другой схемы, вносимые имена должны квалифицироваться именем их родительской схемы.

Пример 51 – Эти EXPRESS-схемы взаимосвязаны, так как схема с именем **primary** использует определение объекта с именем **an_ent** из схемы **secondary**.

```
SCHEMA primary;
   USE FROM secondary (an_ent AS used);

   ENTITY dup;
      att1 : used;
      att2 : BOOLEAN;
   END_ENTITY;
END_SCHEMA;

SCHEMA secondary;

   ENTITY dup;
      name : STRING;
      int  : INTEGER;
   END_ENTITY;

   ENTITY an_ent;
      att3 : dup;
      att4 : REAL;
   END_ENTITY;
END_SCHEMA;
```

Любое использование **an_ent** в экземпляре схемы **primary** требует экземпляра объекта с именем **dup**, который также определяется в схеме **secondary** и автоматически доступен в силу семантики предложения USE. Однако в данном случае в схеме **primary** также имеется объект с именем **dup**. Две их области значений должны различаться внутри EXPRESS-I представления **primary** посредством квалификации имени объекта, вносимого из схемы **secondary,** как показано ниже.

```
MODEL example;
   SCHEMA_DATA primary;
      dup1 = dup{att1 -> @used1; att2 -> TRUE;};
      used1 = used{att3 -> @dup2; att4 -> 1.23;};
      dup2 = secondary.dup{name -> `from secondary`; int -> 1;};
      used2 = used{att3 -> @dup3; att4 -> -3.9;};
   END_SCHEMA_DATA;

   SCHEMA_DATA secondary;
      dup3 = dup{name -> `in secondary`; int -> 3;};
      dup4 = dup{name -> `in secondary`; int -> 4;};
      an_ent1 = an_ent{att3 -> @dup3; att4 -> 42.0;};
   END_SCHEMA_DATA;
END_MODEL;
```

12.2 Отображение простых типов данных из EXPRESS

Отображение простого типа данных из EXPRESS в значение EXPRESS-I задано в таблице 14.

Таблица 14 – Отображение простого типа

EXPRESS	EXPRESS-I
BINARY	BinaryValue
BOOLEAN	BooleanValue
INTEGER	IntegerValue
LOGICAL	LogicalValue
NUMBER	IntegerValue RealValue
REAL	RealValue
STRING	StringValue

Пример 52 – Отображение простых типов данных

EXPRESS

```
ENTITY base;
    a_binary    : BINARY;
    a_boolean   : BOOLEAN;
    an_integer  : INTEGER;
    a_logical   : LOGICAL;
    a_number    : NUMBER;
    a_real      : REAL;
    a_string    : STRING;
END_ENTITY;
```

EXPRESS-I

```
e1 = base {
        a_binary    -> %0110;
        a_boolean   -> FALSE;
        an_integer  -> 12345;
        a_logical   -> UNKNOWN;
        a_number    -> -PI;
        a_real      -> -9.99e2;
        a_string    -> `Tangles`;
      };
```

12.3 Отображение агрегатных типов данных

Отображение агрегаций из EXPRESS в EXPRESS-I приведено в таблице 15.

Таблица 15 – Отображение AGGREGATE

EXPRESS	EXPRESS-I
AGGREGATE ARRAY BAG LIST SET	Одно из следующих: FixedAggr DynamicAggr DynamicAggr DynamicAggr

Отображение “агрегации—агрегации...” производится отображением каждой элементарной агрегации в порядке слева направо. Это значит, что самая левая EXPRESS-агрегация становится самой внешней EXPRESS-I-агрегацией.

Пример 53 – Отображения AGGREGATE

EXPRESS

```
ENTITY aggr;
   an_array  : ARRAY [1:3] OF INTEGER;
   a_bag     : BAG [0:?] OF INTEGER;
   a_list    : LIST [0:2] OF INTEGER;
   a_set     : SET [1:?] OF INTEGER;
   a_mix     : ARRAY [1:2] OF SET OF INTEGER;
END_ENTITY;
```

EXPRESS-I

```
e1 = aggr {
        an_array -> [1, 2, 3];
        a_bag    -> (3, 3, 1);
        a_list   -> (1);
        a_set    -> (9, 5, 11);
        a_mix    -> [(1, 2), (6, 5)];
      };
```

Примечание – EXPRESS ARRAY может иметь значения OPTIONAL. Если значения не определены в экземпляре ARRAY, то эти значения обозначаются в EXPRESS-I конструкцией Nil (то есть символом ?).

Пример 54 – Отображение массива **sparse**

EXPRESS

```
ENTITY sparse;
   a1 : ARRAY [1:4] OF OPTIONAL INTEGER;
   a2 : ARRAY [5:8] OF OPTIONAL INTEGER;
END_ENTITY;
```

EXPRESS-I

```
e1 = sparse {
        a1 -> [1, ?, ?, 4];
        a2 -> [1, ?, 3, ?];
      };
```

12.4 Отображение определенного типа данных из EXPRESS

Определенный тип данных из EXPRESS отображается в EXPRESS-I одним из трех способов:

a) заменой идентификатора EXPRESS-типа значением типа;
b) заменой идентификатора EXPRESS-типа поименованным значением типа;
c) определением экземпляра типа.

Пример 55 – Отображение определенного типа данных

EXPRESS	EXPRESS-I
TYPE dd = ARRAY [1:2] OF INTEGER; END_TYPE;	t3 = dd{[6, 8]};
ENTITY use_type; attr : dd; END_ENTITY;	e1 = use_type{attr -> [2, 4];}; e2 = use_type{attr -> dd{[4, 6]};}; e3 = use_type{attr -> @t3;};

12.5 Отображение перечисляемого типа из EXPRESS

Тип ENUMERATION из EXPRESS отображается в EXPRESS-I одним из трех способов:

a) заменой идентификатора EXPRESS-типа перечисляемым значением;
b) заменой идентификатора EXPRESS-типа поименованным перечисляемым значением;
c) определением экземпляра перечисления.

Пример 56 – Отображение перечисления

EXPRESS	EXPRESS-I
TYPE enum = ENUMERATION OF (one, two, three); END_TYPE;	t3 = enum{!three};
ENTITY use_enum; attr : enum; END_ENTITY;	e1 = use_enum{attr -> !one;}; e2 = use_enum{attr -> enum{!two};}; e3 = use_enum{attr -> @t3;};

12.6 Отображение выбираемого типа из EXPRESS

Тип SELECT из EXPRESS отображается в EXPRESS-I одним из трех способов:

a) заменой идентификатора EXPRESS–типа выбираемым значением;
b) заменой идентификатора EXPRESS–типа поименованным выбираемым значением;
c) определением экземпляра выбора.

EXPRESS-тип SELECT необязательно отображать в EXPRESS-I непосредственно. Детали отображения зависят от того, как формируется тип SELECT, и описаны ниже.

Тип SELECT определяет дерево. Корнем дерева является тип SELECT, а ветви из корня соответствуют типам выбора внутри SELECT. Если одним из этих типов является сам тип SELECT, то он порождает новые ветви и т. д. Листья дерева образуются из выборов, не являющихся типами SELECT. В простом случае все листья являются разными типами. В сложном случае по крайней мере два листа имеют один и тот же базовый тип.

12.6.1 Случай простого выбора

Тип выступает либо как ссылка на один из типов в списке выбора, либо как вхождение одного из типов в список выбора.

Пример 57 – Отображение простого выбора

EXPRESS	EXPRESS-I
ENTITY a; aa : INTEGER; END_ENTITY;	e1 = a{aa -> 3;}; e3 = a{aa -> 9;};
ENTITY b; ab : INTEGER; END_ENTITY;	e2 = b{ab -> 6;}; e4 = b{ab -> 12;};
TYPE s = SELECT(a, b);	s4 = s{@e4};

```
END_TYPE;
ENTITY c;                                   c1 = c{ac -> (@s4, @e3, @e2, @e1);};
   ac : LIST [1:?] OF s;                    c2 = c{ac -> (s{@1}, @e3, @e3);};
END_ENTITY;
```

12.6.2 Случай сложного выбора

В этом случае листья дерева не могут быть различимы только по их значениям. Это происходит, если:

a) листья являются определенными типами данных с идентичными базовыми типами;

b) листья являются типами ENUMERATION, множества значений которых на листьях не разобщены. Например, множества [red, green, blue] и [red, amber, green] не разобщены.

Значение экземпляра выбора в этом случае должно быть представлено в EXPRESS-I либо ссылкой на экземпляр, либо поименованным значением.

Пример 58 – Отображение сложного выбора

EXPRESS	EXPRESS-I
TYPE size = SELECT	s1 = size{@r1};
(area, radius);	s2 = size{radius{4.3}};
END_TYPE;	
TYPE area = REAL;	a1 = area{7.5};
END_TYPE;	
TYPE radius = REAL;	r1 = radius{27.89};
END_TYPE;	
ENTITY circle;	c1 = circle{howbig -> area{PI};};
howbig : size;	c2 = circle{howbig -> radius{1.0};};
WHERE	c3 = circle{howbig -> @s1;};
howbig > 0.0;	c4 = circle{howbig -> @a1};
END_ENTITY;	c5 = circle{howbig -> @s2};

12.7 Отображение EXPRESS-константы

EXPRESS-константа (CONSTANT) синтаксически отображается в EXPRESS-I-конструкцию **constant_spec**. Это значит, что в EXPRESS-I определяются только идентификатор константы и значение, а область значений константы задается в исходном EXPRESS-определении. Кроме того, значение константы должно быть полностью вычисляемым. Определение каждой константы, появляющееся в экземпляре схемы, должно быть объявлено в определении EXPRESS-схемы. Однако не требуется, чтобы каждая EXPRESS-константа присутствовала в экземпляре схемы.

Пример 59 – Отображение констант

EXPRESS	EXPRESS-I
CONSTANT	CONSTANT
zero : NUMBER := 0.0;	zero = = 0.0;
thousand : INTEGER := 1000;	thousand = = 1000;
million : INTEGER := thousand**2;	million = = 1000000;
origin : point := point(0.0, 0.0);	origin = = point{x -> 0.0; y -> 0.0;};
z_axis : vector := [zero, zero, 1.0];	z_axis = = [0.0, 0.0, 1.0];
a_set : SET OF INTEGER := [1, 2, 3*3];	a_set = = (1, 2, 9);
a_bag : BAG OF INTEGER := [1, 3, 1];	
boss : STRING := `sir`;	
underling : STRING := `hey, you`;	underling = = `hey, you`;
END_CONSTANT;	END_CONSTANT;

Заметим, что две константы с именами **a_bag** и **boss** не отображены в данном примере.

12.8 Отображение EXPRESS-объекта

EXPRESS-конструкция объекта (ENTITY) синтаксически отображается в EXPRESS-I-конструкцию экземпляра объекта. Единственными внутренними фрагментами ENTITY, отобра-

жаемыми в EXPRESS-I, являются атрибуты и операторы SUPERTYPE и SUBTYPE, как показано в таблице 16.

Таблица 16 – Обзор отображения ENTITY

EXPRESS	EXPRESS-I
Имя ENTITY	EntityDomain
Оператор SUPERTYPE	BequeathesTo
Оператор SUBTYPE	InheritsFrom
Явный атрибут	RequiredAttr или OptionalAttr
Вычисляемый атрибут	DerivedAttr
Инверсный атрибут	InverseAttr
Оператор UNIQUE	Ничего
Оператор WHERE	Ничего

Пример 60 – Отображение простого объекта

EXPRESS

```
ENTITY top;
   a : SET OF bot;
END_ENTITY;

ENTITY bot;
   i : INTEGER;
DERIVE
   j : INTEGER := 2*i;
INVERSE
   inv : BAG [1:?] OF top FOR a;
UNIQUE
   u1 : i;
WHERE
   w1 : i > 0;
END_ENTITY;
```

EXPRESS-I

```
t1 = top{a -> (@eg1, @eg2);};
t2 = top{a -> (@eg2, @eg3);};
t3 = top{a -> ( );};

eg1 = bot{i -> 1;
          j <- 2;
          inv <- (@t1);};

eg2 = bot{i -> 276;
          j <- 552;
          inv <- (@t1, @t2);};

eg3 = bat{i -> 9876;
          j;
          inv <- (@t2);};
```

12.9 Отображение атрибутов EXPRESS-объекта

EXPRESS-I-атрибуты должны появляться в том же порядке, что и в соответствующем EXPRESS-объекте. Каждый EXPRESS-атрибут должен иметь соответствующий EXPRESS-I-атрибут.

Значение EXPRESS-I-атрибута должно быть совместимо с областью значений EXPRESS-определения.

12.9.1 Явный атрибут

Явные EXPRESS-атрибуты отображаются непосредственно в EXPRESS-I-атрибуты. Описание EXPRESS-атрибута повторяется в EXPRESS-I, за исключением того, что описание типа атрибута (то есть справа от двоеточия) заменяется значением типа атрибута, а двоеточие заменяется на ->.

Значение может быть представлено простым значением, ссылкой на экземпляр предмета (то есть ссылкой на экземпляр объекта, типа, перечисления или выбора), значением перечисления, поименованным значением, ссылкой на константу, ссылкой на параметр или агрегатами данных значений. Эти значения более детально обсуждены ниже.

В случае, если явный атрибут является необязательным (OPTIONAL), значением атрибута может быть также Nil, показывающее, что значение не представлено.

Пример 61 – Отображение необязательного атрибута

EXPRESS

```
ENTITY opt;
   req     : STRING;
   opt_att : OPTIONAL REAL;
END_ENTITY;
```

EXPRESS-I

```
opt1 = opt{req -> `Opt_att given`;
           opt_att -> 5.0;};

opt2 = opt{req -> `Opt_att not given`;
           opt_att -> ?;};
```

Примечание – В EXPRESS-I явный атрибут может иметь значение Nil; в этом случае экземпляр не соответствует EXPRESS-определению.

12.9.2 Вычисляемые и инверсные атрибуты

Вычисляемые EXPRESS-атрибуты отображаются в EXPRESS-I аналогично явным атрибутам, за исключением того, что двоеточие заменяется знаком <-.

Инверсные EXPRESS-атрибуты отображаются в EXPRESS-I аналогично явным атрибутам, за исключением того, что двоеточие заменяется знаком <-, а значением атрибута является динамическая агрегация ссылок на экземпляр объекта.

Не требуется, чтобы в EXPRESS-I присутствовали значения вычисляемых или инверсных атрибутов, хотя имена ролей присутствовать должны.

Примечания

1 По определению, значение вычисляемого атрибута может быть определено по значениям явных атрибутов. Аналогично, значение инверсного атрибута экземпляра объекта может быть определено по значениям атрибутов экземпляров других объектов, которые ссылаются на экземпляр объекта с данным инверсным атрибутом. Таким образом, по крайней мере, концептуально, значения как вычисляемого, так и инверсного атрибутов являются вычислимыми свойствами.

2 С другой стороны, значения явных атрибутов являются базовыми входными данными, не вычислимыми внутри системы EXPRESS-I.

3 Символы -> и <- были выбраны для индикации этой разницы в качестве значений атрибута.

12.9.3 Атрибут с простой областью значений

Если областью значений EXPRESS-атрибута является простой тип данных, это должно быть отображено как значение EXPRESS-I, принадлежащее простой области значений. Обычно это является простым значением, но может быть ссылкой на константу или параметр, областями значений которых являются простые области значений.

Правила и ограничения

a) Ссылка на константу должна использоваться только в случае, если и экземпляр объекта, и экземпляр константы находятся внутри одного и того же экземпляра данных схемы.

b) Ссылка на параметр должна использоваться только в случае, если формальный параметр и экземпляр объекта находятся внутри одного и того же контекста (CONTEXT).

c) Ссылка на параметр не должна использоваться внутри области действия MODEL.

Пример 62 – Отображение простого значения в качестве атрибута

Пусть дано EXPRESS-представление в виде:

```
SCHEMA a_schema;
    CONSTANT
        const : INTEGER:=275;
    END_CONSTANT;

    ENTITY an_ent;
        aa : INTEGER;
    END_ENTITY;
END_SCHEMA;
```

Тогда EXPRESS-I-представление может иметь вид:

```
MODEL some_data;
    SCHEMA_DATA a_schema;
        CONSTANT
            const == 275;
        END_CONSTANT;

        a1 = an_ent{aa -> 1;};
        a2 = an_ent{aa -> const;};
        a3 = an_ent{aa -> 21;};
        a4 = an_ent{aa -> 987;};
    END_SCHEMA_DATA;
END_MODEL
```

Это можно представить и по-другому, через контекст:

```
CONTEXT  a_context;
      PARAMETER
            parm1  :  INTEGER  :=  21;
            parm2  :  INTEGER  :=  987;
      END_PARAMETER;

      SCHEMA_DATA a_schema;

            CONSTANT
               const  =  =  275;
            END_CONSTANT;

            a1  =  an_ent{aa -> 1;};
            a2  =  an_ent{aa -> const;};
            a3  =  an_ent{aa -> param1;};
            a4  =  an_ent{aa -> param2;};
       END_SCHEMA_DATA;
END_CONTEXT;
```

12.9.4 Атрибут с областью значений объекта

Если областью значений EXPRESS-атрибута является объект, то атрибут должен отображаться в значение EXPRESS-I, принадлежащие к области значений объекта. Обычно это является ссылкой на экземпляр объекта, но может быть ссылкой на константу или на параметр, областью значений которого является область значений объекта.

Правила и ограничения

a) Ссылка на константу может использоваться только в случае, если экземпляр объекта и экземпляр константы находятся в одном и том же экземпляре данных схемы.

b) Ссылка на параметр может использоваться только в случае, если формальный параметр и экземпляр объекта находятся в одном и том же контексте (CONTEXT).

c) Ссылка на параметр не должна использоваться в области применения модели (*MODEL*).

d) Ни ссылка на параметр, ни ссылка на константу не должны использоваться для инверсного атрибута.

Пример 63 – Отображение объекта в качестве атрибута

Пусть EXPRESS-представление задано в виде:

```
SCHEMA  a_schema;
      CONSTANT
         const  :  an_ent  :=  an_ent(275);
      END_CONSTANT;

      ENTITY  an_ent;
         aa  :  INTEGER;
      END_ENTITY;

      ENTITY  bdyn;
      ab  :  an_ent;
      END_ENTITY;
END_SCHEMA;
```

Тогда EXPRESS-I-представление может иметь вид:

```
CONTEXT  a_context;
            PARAMETER
               param  :  an_ent  :=  an_ent{aa -> 42;};
            END_PARAMETER;

      SCHEMA_DATA a_schema;
```

```
        CONSTANT
           const  = =  an_ent{aa -> 275;};
        END_CONSTANT;

        a1  =  an_ent{aa -> 1;};
        b1  =  bdyn{ab -> @a1;};
        b2  =  bdyn{ab -> const;};
        b3  =  bdyn{ab -> param;};
    END_SCHEMA_DATA;
END_CONTEXT;
```

12.9.5 Атрибут с областью значений типа, выбора или перечисления

Если областью значений EXPRESS-атрибута является определенный тип данных, типы SELECT или ENUMERATION, то атрибут должен отображаться как значение EXPRESS-I, принадлежащее к соответствующей области значений. Обычно это либо значение (для определенного типа данных или перечисления), либо ссылка на экземпляр объекта (для выбора), но может быть и ссылкой на экземпляр предмета, поименованным значением, либо ссылкой на константу или параметр, области значений которых совместимы с областью значений атрибута.

Правила и ограничения

a) Ссылка на константу может использоваться только в случае, если экземпляр объекта и экземпляр константы находятся в одном и том же экземпляре данных схемы.

b) Ссылка на параметр может использоваться только в случае, если формальный параметр и экземпляр объекта находятся в одном и том же контексте (CONTEXT).

c) Ссылка на параметр не должна использоваться внутри области применения модели (MODEL).

d) Ссылка на экземпляр предмета или поименованное значение должны использоваться, если фактическая область значений не однозначно определяется из значения.

Пример 64 – Отображение типов в качестве атрибутов

Пусть EXPRESS-представление задано в виде:

```
SCHEMA  a_schema;
        CONSTANT
           zero  :  REAL  :=  0.0;
        END_CONSTANT;

        TYPE  size  =  SELECT(area, radius); END_TYPE;
        TYPE  area  =  REAL; END_TYPE;
        TYPE  radius  =  REAL; END_TYPE;
        TYPE  vector  =  ARRAY [1:3] OF REAL; END_TYPE;
        TYPE  color  =  ENUMERATION OF (red, blue, green); END_TYPE;

        ENTITY  point;
           x, y, z,  :  REAL;
        END_ENTITY;

        ENTITY  circle;
           center   :  point;
           normal  :  vector;
           howbig:   size;
           shade            :  color;
        END_ENTITY;
END_SCHEMA;
```

Тогда EXPRESS-I-представление может иметь вид:

```
SCHEMA_DATA  a_schema;

        CONSTANT
           zero  = =  0. 0;
        END_CONSTANT;
```

```
unit_rad  =  size{radius {1.0}};
x_axis  =  vector{[1.0, zero, zero ]};
z_axis  =  vector{[zero, zero, 1.0]};
x_color  =  color{!red};

p0  =  point{x -> zero; y -> zero, z -> zero;};
p1  =  point{x -> 1.0; y -> 1.0, z -> 1.0;};

c1  =  circle{center -> @p0;
              normal -> @x_axis;
              howbig -> area{PI};
              shade -> @x_color;);
c2  =  circle{center -> @p0;
              normal -> [1.0, 2.0, 3.0];
              howbig -> radius{33.0};
              shade -> !blue;};
c3  =  circle{center -> @p1;
              normal -> @z_axis;
              howbig -> @unit_rad;
              shade -> !blue;};
END_SCHEMA_DATA;
```

12.10 Отображение супертипов и подтипов

Имеется взаимно однозначное соответствие между супертипами и подтипами EXPRESS и супертипами и подтипами EXPRESS-I (см. таблицу 17).

Таблица 17 – Обзор отображения SUPERTYPE и SUBTYPE

EXPRESS	EXPRESS-I
SUPERTYPE OF (...) SUBTYPE OF (...)	BequeathesTo InheritsFrom

В EXPRESS-I наполнение объекта, являющегося листом дерева супертипов/подтипов, требует наполнения всех его супертипов. Дерево экземпляра супертипов EXPRESS-I должно быть всегда выписано полностью.

Пример 65 – Рассмотрим ниже фрагмент дерева EXPRESS и конкретный объект **me**:

```
ENTITY  ...
ENTITY  parent SUBTYPE OF (grandparent)
                SUPERTYPE OF (me ANDOR sibbing );
                ...
ENTITY  me      SUBTYPE OF (parent)
                SUPERTYPE OF (elder ANDOR younger);
                ...
ENTITY  elder   SUBTYPE OF (me)
                SUPERTYPE OF ...
ENTITY  ...
```

Объект **me** наследует любые атрибуты, которые могут иметь его супертипы (то есть **parent**, **grandparent** и т.д.). В свою очередь, **me** завещает как свои наследуемые атрибуты, так и свои собственные атрибуты своим подтипам (то есть **elder**, **younger** и их последующих потомков).

В этом дереве экземпляр **me** может также иметь либо не иметь **sibling** (брата). В общем дереве возможно существование многих отношений, не находящихся на прямой линии предка и потомка.

Для данного подраздела определим:

экземпляр прямого дерева (direct tree instance): Экземпляр однокоренного дерева подтипов/супертипов, имеющего единственный путь, при незаполненных ветвях, от корня к единственному листу;

экземпляр общего дерева (general tree instance): Экземпляр дерева подтипов/супертипов, не являющийся экземпляром прямого дерева.

Дерево EXPRESS, в котором все отношения SUPERTYPE являются ONEOF и ни один из SUBTYPE не имеет множественных SUPERTYPE, всегда является прямым деревом.

Наполнение дерева, включающего отношения ANDOR, будет прямым, если все отношения ANDOR наполняются как отношения ONEOF; в противном случае по крайней мере некоторая часть наполняемого дерева не будет прямой. Наполнение отношений AND всегда дает общее дерево. Наполнение объекта, имеющего множественные SUPERTYPE, всегда дает общее дерево.

В экземпляре прямого дерева должен быть представлен полный путь экземпляра от корня к листу.

Следующий набор правил определяет отображение общего дерева:

a) полный путь экземпляра от корня к месту, включающий боковые ветви, всегда должен наполняться в соответствии с приведенными ниже правилами;

b) если наполняемый объект (ENTITY) является SUBTYPE для одного или более объектов, то каждый из SUPERTYPE объектов должен наполняться;

c) если наполняемый объект (ENTITY) является SUPERTYPE для одного или более объектов (то есть имеется отношение AND либо имеется отношение ANDOR, которое наполняется, скорее, как AND, нежели как ONEOF), то SUPERTYPE и все его одновременно существующие SUBTYPE должны наполняться;

d) если SUPERTYPE объекта (ENTITY) помечен как ABSTRACT, то экземпляр этого объекта будет иметь по крайней мере один экземпляр SUBTYPE. Если SUPERTYPE не помечен как ABSTRACT, то он может иметь либо не иметь экземпляров SUBTYPE, в зависимости от конкретных данных.

Примечание – Упорядочение экземпляров объектов в дереве подтипов/супертипов не имеет значения.

Пример 66 – Отображение дерева

Пусть дано следующее EXPRESS-представление:

```
ENTITY root;
    g_name : STRING;
END_ENTITY;

ENTITY node
    SUBTYPE OF (root);
    p_name : STRING;
END_ENTITY;

ENTITY leaf1
    SUBTYPE OF (node);
    my_name : STRING;
END_ENTITY;

ENTITY leaf2
    SUBTYPE OF (node);
    s_name : STRING;
END_ENTITY;
```

Тогда двумя примерами экземпляров этой структуры могут быть:

ЭКЗЕМПЛЯР 1

```
c1[1] = root{
          g_name -> `root`;
          SUPOF(@2);};

c1[2] = node{
          SUBOF(@1);
          p_name -> `trunk`;
          SUPOF(@3, @4);};

c1[3] = leaf1{
        SUBOF(@2);
```

ЭКЗЕМПЛЯР 2

```
c2[1] = root{
          g_name ->`base`;
          SUPOF(@2);};

c2[2] = node{
          SUBOF(@1);
          p_name ->`branch`;
          SUPOF(@3);};

c2[3] = leaf1{
        SUBOF(@2);
```

```
        my_name -> `self`;};                    my_name ->`twig`;};
c1[4] = leaf2{
            SUBOF(@2);
            s_name ->`sibling`;};
```

Экземпляр, помеченный 1, является экземпляром общего дерева, а экземпляр, помеченный 2, – экземпляром прямого дерева.

12.10.1 Отображение переобъявляемых атрибутов

В подтипе EXPRESS есть возможность переобъявлять атрибуты, наследуемые от супертипа. В EXPRESS-I переобъявление выступает как ограничение значения атрибута. Переобъявляемые атрибуты не должны именоваться внутри экземпляра подтипа.

Пример 67 – В примере объект **real_point** является подтипом **point** и переобъявляет его атрибуты типа NUMBER на тип REAL. Имеются два соответствующих экземпляра EXPRESS-I. Первый (то есть **p1**) является экземпляром простого объекта только супертипа и отображает значения атрибутов как тип NUMBER. Второй (то есть **p2**) является экземпляром сложного объекта, где **p2[1]** является компонентом супертипа, а **p2[2]** - компонентом подтипа. В подтипе не показаны атрибуты, но значения отражаемые в супертипе, ограничены типом REAL.

EXPRESS

```
ENTITY point;
    x : NUMBER;
    y : NUMBER;
END_ENTITY;

ENTITY real_point;
    SUBTYPE OF (point);
    SELF\point.x : REAL
    SELF\point.y : REAL;
END_ENTITY;
```

EXPRESS-I

```
p1 = point{x -> 1;
           y -> 2;};

p2[1] = point{x -> 1.5;
              y -> 2.7;
              SUPOF(@2);};

p2[2] = real_point{SUBOF(@1);};
```

В случае переобъявления наследуемого явного атрибута на вычисляемый атрибут переобъявленный атрибут должен выступать в супертипе как вычисляемый атрибут, когда бы ни наполнялся переобъявляющий подтип.

Пример 68 – Следующее EXPRESS-представление объявляет **circle** как окружность, определяемую центром и радиусом. Объект **circle_2**pt является разновидностью **circle**, определяемой центром и точкой на окружности **circle**. Наследуемый атрибут **radius** переобъявляется как вычисляемый атрибут, значение которого задается расстоянием между двумя точками.

```
ENTITY circle;
    centre : point;
    radius : REAL;
END_ENTITY;

ENTITY circle_2pt
    SUBTYPE OF (circle);
    circum_pnt : point;
DERIVE
    SELF\circle.radius : REAL := distance(SELF\circle.center, circum_pnt);
END_ENTITY;
```

В EXPRESS-I-представлении экземплярами **circle** и **circle_2pt** могут быть:

```
c1 = circle{centre -> [1.0, 0.0];
            radius -> 2.0;};

c2pt[21] = circle{centre -> [1.0, 0.0];
                  radius <- 2.0;
                  SUPOF (@5);};

c2pt[5] = circle_2pt{SUBOF(@21);
                     circum_pnt -> [1.0, 2.0];};
```

ПРИЛОЖЕНИЕ А
(обязательное)

Описание синтаксиса EXPRESS-I

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

Примечания

1 Многие элементы языка EXPRESS доступны для использования при определении контрольных примеров. Недоступные для использования элементы EXPRESS касаются определения EXPRESS-схем, интерфейсов схем и правил. Для удобства читателя элементы EXPRESS приведены в настоящем приложении со справочными примечаниями. Для полноты описания языка также в виде комментариев приведены правила, относящиеся к недоступным для использования элементам EXPRESS.

2 В качестве дальнейшего ориентира в конструкциях, относящиеся только к EXPRESS-I, подчеркивание не используется – каждое имя в конструкции EXPRESS-I начинается с заглавной буквы. Например, **DerivedAttr** является конструкцией EXPRESS-I, тогда как **derived_attr** является конструкцией EXPRESS. Кроме того, исходная нумерация правил EXPRESS оставлена без изменений. Специфические правила EXPRESS-I пронумерованы с добавлением символа 'i'.

3 Приведенное определение синтаксиса, понимаемое буквально, будет вызывать неодназначное толкование у анализаторов. Оно написано для представления информации, относящейся к использованию идентификаторов. Интерпретируемые идентификаторы определяют лексемы, являющиеся ссылками на объявляемые идентификаторы, и поэтому не разрешены в **simple_id**. Это требует от разработчика синтаксического анализатора представления таблицы поиска или аналогичной конструкции, позволяющей разрешать ссылку на идентификатор и возвращать лексему требуемой ссылки контролеру грамматических правил. Этот подход использован для того, чтобы помочь разработчикам синтаксических анализаторов в устранении неоднозначностей, относящихся к использованию идентификаторов.

A.1 Лексемы

Последующие правила определяют лексемы, используемые в EXPRESS-I. Внутри текста, соответствующего отдельному синтаксическому правилу следующих подразделов и разделов: A1.1, A1.2, A.2 и A3, не должны присутствовать пробелы или примечания, за исключением случаев, оговоренных в синтаксических правилах.

A.1.1 Ключевые слова

В данном подразделе приведены правила, используемые для представления ключевых слов EXPRESS-I.

Примечание – В данном подразделе придерживаются типографского соглашения, по которому каждое ключевое слово представляется синтаксическим правилом, левой частью которого является данное ключевое слово, записанное заглавными буквами. Правило 15i является исключением, обусловленным необходимостью избежать пересечения с правилом 251. Так как строковые литералы в синтаксических правилах не чувствительны к типу буквы, ключевые слова могут быть записаны в исходном коде на EXPRESS-I заглавными, строчными либо смешанного (того и другого) типа буквами.

```
0i  CALL = 'call' .
1i  CRITERIA = 'criteria' .
2i  END_CALL = 'end_call' .
3i  END_CRITERIA = 'end_criteria' .
4i  END_NOTES = 'end_notes' .
5i  END_OBJECTIVE = 'end_objective' .
6i  END_PARAMETER = 'end_parameter' .
7i  END_PURPOSE = 'end_purpose' .
8i  END_REALIZATION = 'end_realization' .
9i  END_REFERENCES = 'end_references' .

10i END_SCHEMA_DATA = 'end_schema_data' .
11i END_TEST_CASE = 'end_test_case' .
12i IMPORT = 'import' .
13i NOTES = 'notes' .
14i OBJECTIVE = 'objective' .
15i PARAMETERi = 'parameter' .
16i PURPOSE = 'purpose' .
17i REALIZATION = 'realization' .
18i REFERENCES = 'references' .
```

```
19i SCHEMA_DATA = 'schema_data' .

20i SUBOF = 'subof' .
21i SUPOF = 'supof' .
22i TEST_CASE = 'test_case' .
23i USING = 'using' .
24i WITH = 'with' .
```

Примечание — Следующие EXPRESS-правила от 0 до 118, за исключением 8, 37, 38, 49, 84, 89, 90 и 110, используются в EXPRESS-I.

```
 0 ABS = 'abs' .
 1 ABSTRACT = 'abstract' .
 2 ACOS = 'acos' .
 3 AGGREGATE = 'aggregate' .
 4 ALIAS = 'alias' .
 5 AND = 'and' .
 6 ANDOR = 'andor' .
 7 ARRAY = 'array' .
< 8 AS = 'as' . >
 9 ASIN = 'asin' .

10 ATAN = 'atan' .
11 BAG = 'bag' .
12 BEGIN = 'begin' .
13 BINARY = 'binary' .
14 BLENGTH = 'blength' .
15 BOOLEAN = 'boolean' .
16 BY = 'by' .
17 CASE = 'case' .
18 CONSTANT = 'constant' .
19 CONST_E = 'const_e' .

20 CONTEXT = 'context' .
21 COS = 'cos' .
22 DERIVE = 'derive' .
23 DIV = 'div' .
24 ELSE = 'else' .
25 END = 'end' .
26 END_ALIAS = 'end_alias' .
27 END_CASE = 'end_case' .
28 END_CONSTANT = 'end_constant' .
29 END_CONTEXT = 'end_context' .

30 END_ENTITY = 'end_entity' .
31 END_FUNCTION = 'end_function' .
32 END_IF = 'end_if' .
33 END_LOCAL = 'end_local' .
34 END_MODEL = 'end_model' .
35 END_PROCEDURE = 'end_procedure' .
36 END_REPEAT = 'end_repeat' .
< 37 END_RULE = 'end_rule' . >
< 38 END_SCHEMA = 'end_schema' . >
39 END_TYPE = 'end_type' .

40 ENTITY = 'entity' .
41 ENUMERATION = 'enumeration' .
42 ESCAPE = 'escape' .
43 EXISTS = 'exists' .
44 EXP = 'exp' .
45 FALSE = 'false' .
46 FIXED = 'fixed' .
```

```
47  FOR = 'for' .
48  FORMAT = 'format' .
<  49  FROM = 'from' . >

50  FUNCTION = 'function' .
51  GENERIC = 'generic' .
52  HIBOUND = 'hibound' .
53  HIINDEX = 'hiindex' .
54  IF = 'if' .
55  IN = 'in' .
56  INSERT = 'insert' .
57  INTEGER = 'integer' .
58  INVERSE = 'inverse' .
59  LENGTH = 'length' .

60  LIKE = 'like' .
61  LIST = 'list' .
62  LOBOUND = 'lobound' .

63  LOCAL = 'local' .
64  LOG = 'log' .
65  LOG10 = 'log10' .
66  LOG2 = 'log2' .
67  LOGICAL = 'logical' .
68  LOINDEX = 'loindex' .
69  MOD = 'mod' .

70  MODEL = 'model' .
71  NOT = 'not' .
72  NUMBER = 'number' .
73  NVL = 'nvl' .
74  ODD = 'odd' .
75  OF = 'of' .
76  ONEOF = 'oneof' .
77  OPTIONAL = 'optional' .
78  OR = 'or' .
79  OTHERWISE = 'otherwise' .

80  PI = 'pi' .
81  PROCEDURE = 'procedure' .
82  QUERY = 'query' .
83  REAL = 'real' .
<  84  REFERENCE = 'reference' .
85  REMOVE = 'remove' .
86  REPEAT = 'repeat' .
87  RETURN = 'return' .
88  ROLESOF = 'rolesof' .
<  89  RULE = 'rule' . >

<  90  SCHEMA = 'schema' . >

91  SELECT = 'select' .
92  SELF = 'self' .
93  SET = 'set' .
94  SIN = 'sin' .
95  SIZEOF = 'sizeof' .
96  SKIP = 'skip' .
97  SQRT = 'sqrt' .
98  STRING = 'string' .
99  SUBTYPE = 'subtype' .

100  SUPERTYPE = 'supertype' .
101  TAN = 'tan' .
```

```
102 THEN = 'then' .
103 TO = 'to' .
104 TRUE = 'true' .
105 TYPE = 'type' .
106 TYPEOF = 'typeof' .
107 UNIQUE = 'unique' .
108 UNKNOWN = 'unknown' .
109 UNTIL = 'until' .

< 110 USE = 'use' . >
111 USEDIN = 'usedin' .
112 VALUE = 'value' .
113 VALUE_IN = 'value_in' .
114 VALUE_UNIQUE = 'value_unique' .
115 VAR = 'var' .
116 WHERE = 'where' .
117 WHILE = 'while' .
118 XOR = 'xor' .
```

A.1.2. Классы символов

Следующие правила определяют различные классы символов, используемые при конструировании лексем в А2.

Примечание – Последующие правила EXPRESS от 119 до 135 используются в EXPRESS-I.

```
119 bit = '0' | '1' .

120 digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' .
121 digits = digit { digit } .
122 encoded_character = octet octet octet octet .
123 hex_digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' .
124 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' .
125 lparen_not_star = '(' not_star .
126 not_lparen_star = not_paren_star | ')' .
127 not_paren_star = letter | digit | not_paren_star_special .
128 not_paren_star_quote_special = '!' | '»' | '#' | '$' | '%' | '&' | '+' | ',' |
                                 '-' | '.' | '/' | ':' | ';' | '<' | '=' | '>' |
                                 '?' | '@' | '[' | '\' | ']' | '^' | '_' | '`' |
                                 '{' | '|' | '}' | '~' .
129 not_paren_star_special = not_paren_star_quote_special | '"' .

130 not_quote = not_paren_star_quote_special | letter | digit | '(' | ')' | '*' .
131 not_rparen = not_paren_star | '*' | '(' .
132 not_star = not_paren_star | '(' | ')' .
133 octet = hex_digit hex_digit .
134 special = not_paren_star_quote_special | '(' | ')' | '*' | '"' .
135 star_not_rparen = '*' not_rparen .
```

А.2 Лексические элементы

Следующие правила устанавливают, как определенные комбинации символов интерпретируются в качестве лексических элементов языка.

```
25i BinaryValue = binary_literal .
26i Description = { \a | \s | \n } .
27i EncodedStringValue = '»' { encoded_character | \n } '"' .
28i EnumerationValue = '!' simple_id .
29i IntegerValue = [ sign ] integer_literal .

30i Nil = '?' .
31i SignedMathConstant = [ sign ] MathConstant .
32i SignedRealLiteral = [ sign ] real_literal .
33i SimpleStringValue = \q { ( \q \q ) | not_quote | \s | \o | \n } \q .
```

Примечание – Следующие EXPRESS-правила 136—141 используются в EXPRESS-I.

```
136 binary_literal = '%' bit { bit } .
137 encoded_string_literal = '»' encoded_character { encoded_character } '"' .
138 integer_literal = digits .
139 real_literal = digits '.' [ digits ] [ 'e' [ sign ] digits ] .

140 simple_id = letter { letter | digit | '_' } .
141 simple_string_literal = \q { ( \q \q ) | not_quote | \s | \x8 | \x9 | \xA
                          | \xB | \xC | \xD } \q .
```

A.2.1 Примечания

Следующие правила устанавливают синтаксис примечаний в EXPRESS-I.

Примечание – Следующие EXPRESS-правила 142—144 используются в EXPRESS-I.

```
142 embedded_remark = '(*' { not_lparen_star | lparen_not_star |
                          star_not_lparen | embedded_remark } '*)' .
143 remark = embedded_remark | tail_remark .
144 tail_remark = '--' { \a | \s | \x8 | \x9 | \xA | \xB | \xC | \xD } \n .
```

A.3 Интерпретируемые идентификаторы

Следующие правила определяют идентификаторы, наделенные некоторым специальным смыслом (например, объявленные где-либо как типы, функции и т. д.).

Примечание – Предполагается, что идентификаторы, соответствующие этим синтаксическим правилам, известны реализации. Каким образом реализация получает эту информацию, не является предметом определения языка. Одним из методов получения этой информации являются многопроходный анализ: первый проход собирает идентификаторы из их объявлений, так что последующие проходы позволяют отличать, например, **variable_ref** от **function_ref**.

```
34i ComplexEntityInstanceRef = '@' SimpleEntityInstanceId .
35i ConstantRef = ConstantId .
36i ContextRef = ContextId .
37i EntityInstanceRef = ComplexEntityInstanceRef | SimpleEntityInstanceRef .
38i EnumerationInstanceRef = '@' EnumerationInstanceId .
39i ParameterRef = ParameterId .

40i SelectInstanceRef = '@' SelectInstanceId .
41i SimpleInstanceRef = '@' SimpleInstanceId .
42i SimpleEntityInstanceRef = '@' SimpleEntityInstanceId .
43i SupSubRef = '@' SupSubId .
44i TypeInstanceRef = '@' TypeInstanceId .
```

Примечание – Следующие EXPRESS-правила 145-155 используются в EXPRESS-I.

```
145 attribute_ref = attribute_id .
146 constant_ref = constant_id .
147 entity_ref = entity_id .
148 enumeration_ref = enumeration_id .
149 function_ref = function_id .

150 parameter_ref = parameter_id .
151 procedure_ref = procedure_id .
152 schema_ref = schema_id .
153 type_label_ref = type_label_id .
154 type_ref = type_id .
155 variable_ref = variable_id .
```

A.4 Грамматические правила

Следующие правила устанавливают, как описанные выше лексические элементы можно комбинировать в конструкции EXPRESS-I. Пробелы и (или) примечания могут появляться между любыми двумя лексемами этих правил. Первичным синтаксическим правилом для EXPRESS-I служит **ExpressISyntax**.

```
45i AcualParameter = ParametrRef ':=' ParmValue ';' .
46i AggregationValue = DynamicAggr | FixedAggr .
47i Assignment = variable_id ':=' SelectableInstanceRef ';' .
```

```
48i  BaseValue = EnumerationValue | SimpleValue .
49i  BequeathesTo = SUPOF DynamicSupSubRefList ';' .

50i  BooleanValue = TRUE | FALSE .
51i  ComplexEntityInstanceId = SimpleEntityInstanceId '[' SupSubId ']' .
52i  ConstantBlock = CONSTANT { ConstantSpec } END_CONSTANT ';' .
53i  ConstantId = constant_ref .
54i  ConstantSpec = ConstantId '= =' ConstantValue ';' .
55i  ConstantValue = AggregationValue | BaseValue | EntityInstanceValue |
                    NamedInstanceValue | SelectValue | TypeValue .
56i  ContextBlock = CONTEXT ContextId ';' ContextBody END_CONTEXT ';' .
57i  ContextBody = { SchemaReferenceSpec } [ FormalParameterBlock ]
                   { SchemaInstanceBlock | SupportAlgorithm } .
58i  ContextId = simple_id .
59i  DerattValue = AggregationValue | BaseValue | EntityInstanceRef |
                   EntityInstanceValue | EnumerationInstanceValue |
                   TypeInstanceRef | TypeInstanceValue | TypeValue .

60i  DerivedAttr = RoleName [ '<-' DerattValue ] ';' .
61i  DynamicAggr = '(' [ DynamicList ] ')' .
62i  DynamicEntityRefList = '(' [ EntityRefList ] ')' .
63i  DynamicList = DynamicMember { ',' DynamicMember } .
64i  DynamicMember = AggregationValue | ConstantValue | DerattValue |
                    ParmValue | ReqattValue | TypeValue .
65i  DynamicSupSubList = '(' [ SupSubRef { ',' SupSubRef } ] ')' .
66i  EntityDomain = [ SchemaId '.' ] EntityId .
67i EntityId = entity_ref .
68i  EntityInstance = EntityInstanceId '=' EntityInstanceValue ';' .
69i  EntityInstanceId = ComplexEntityInstanceId | SimpleEntityInstanceId .

70i  EntityInstanceValue = EntityDomain '{'
                           [ InheritsFrom ]
                           { ExplicitAttr }
                           { DerivedAttr }
                           { InverseAttr }
                           [ BequeathesTo ] '}' .
71i  EntityRefList = EntityInstanceRef { ',' EntityInstanceRef } .
72i  EnumerationDomain = [ SchemaId '.' ] EnumerationId .
73i  EnumerationId = type_ref .
74i  EnumerationInstance = EnumerationInstanceId '='
                           EnumerationInstanceValue ';' .
75i  EnumerationInstanceId = simple_id .
76i  EnumerationInstanceValue = EnumerationDomain
                                '{' EnumerationValue '}' .
77i  ExplicitAttr = RequiredAttr | OptionalAttr .
78i  ExpressISyntax = { TestCaseBlock } { ContextBlock } { ModelBlock }
                      { SchemaInstanceBlock } { ObjectInstance } .
79i  FixedAggr = '[' FixedList ']' .

80i  FixedList = FixedMember { ',' FixedMember } .
81i  FixedMember = DynamicMember | Nil .
82i  FormalParameter = ParameterId ':' parameter_type
                       [ ':=' ParmValueDefault ] ';' .
83i  FormalParameterBlock = PARAMETERi { FormalParameter }
                            END_PARAMETER ';' .
84i  ImportSpec = IMPORT '(' { Assignment } ')' ';' .
85i  InheritsFrom = SUBOF DynamicSupSubRefList ';' .
86i  InvattValue = DynamicEntityRefList .
87i  InverseAttr = RoleName [ '<-' InvattValue ] ';' .
88i  LogicalValue = logical_literal .
89i  MathConstant = CONST_E | PI .
```

```
90i  ModelBlock = MODEL ModelId ';' ModelBody END_MODEL ';' .
91i  ModelBody = { SchemaInstanceBlock } .
92i  ModelId = simple_id .
93i  NamedInstanceValue = EnumerationInstanceValue | SelectInstanceValue |
                          TypeInstanceValue .
94i  NumberValue = IntegerValue | RealValue .
95i  ObjectInstance = EntityInstance | EnumerationInstance | SelectInstance |
                      TypeInstance | SimpleInstance .
96i  ObjectInstanceRef = EntityInstanceRef | EnumerationInstanceRef |
                         SelectInstanceRef | TypeInstanceRef |
                         SimpleInstanceRef .
97i  ObjectiveBlock = OBJECTIVE { TestPurpose } { TestReference }
                      { TestCriteria } { TestNotes } END_OBJECTIVE ';' .
98i  OptattValue = ReqattValue | Nil .
99i  OptionalAttr = RoleName '->' OptattValue ';' .

100i ParameterId = simple_id .
101i ParameterSpec = WITH '(' { ActualParameter } ')' ';' .
102i ParmValue = ObjectInstanceRef | expression .
103i ParmValueDefault = AggregationValue | BaseValue | ConstantRef |
                       EntityInstanceValue | NamedInstanceValue |
                       ObjectInstanceRef | SelectValue | TypeValue |
                       expression .
104i RealValue = SignedMathConstant | SignedRealLiteral .
105i ReqattValue = AggregationValue | BaseValue | ConstantRef |
                   NamedInstanceValue | ObjectInstanceRef | ParameterRef |
                   SelectValue | TypeValue | .
106i RequiredAttr = RoleName '->' ( ReqattValue | Nil ) ';' /
107i RoleName = attribute_ref .
108i SchemaId = schema_ref .
109i SchemaInstanceBlock = SCHEMA_DATA SchemaId ';'
                          [ SchemaInstanceBody ] END_SCHEMA_DATA ';' .

110i SchemaInstanceBody = [ ConstantBlock ] { ObjectInstance } .
111i SchemaReferences = SchemaReferenceSpec { SchemaReferenceSpec } .
112i SchemaReferenceSpec = WITH schema_ref [ USING '(' resource_ref
                          { ',' resource_ref } ')' ] ';' .
113i SelectableInstanceRef = EntityInstanceRef | EnumerationInstanceRef |
                            SelectInstanceRef | TypeInstanceRef .
114i SelectDomain = [ SchemaId '.' ] SelectId .
115i SelectId = type_ref .
116i SelectInstance = SelectInstanceId '=' SelectInstanceValue ';' .
117i SelectInstanceId = simple_id .
118i SelectInstanceValue = SelectDomain '{' SelectValue '}' .
119i SelectValue = EnumerationValue | NamedInstanceValue |
                 ObjectInstanceRef | TypeValue .

120i SimpleEntityInstanceId = simple_id .
121i SimpleInstance = SimpleInstanceId '=' SimpleValue ';' .
122i SimpleInstanceId = simple_id .
123i SimpleValue = BinaryValue | BooleanValue | LogicalValue |
                 NumberValue | StringValue .
124i StringValue = Simple StringValue | EncodedStringValue .
125i SupSubId = digits .
126i SupportAlgorithm = function_decl | procedure_decl .
127i TestCaseBlock = TEST_CASE TestCaseId ';'
                   TestCaseBody END_TEST_CASE ';' .
128i TestCaseBody = SchemaReferences ObjectiveBlock TestRealization
                   { SupportAlgorithm } .
129i TestCaseId = simple_id .
```

```
130i TestRealization = REALIZATION { local_decl } { UseContextBlock }
                       { assignment_stmt } END_REALIZATION ';' .
131i TestCriteria = CRITERIA Description END_CRITERIA ';' .
132i TestNotes = NOTES Description END_NOTES ';' .
133i TestPurpose = PURPOSE Description END_PURPOSE ';' .
134i TestReference = REFERENCE Description END_REFERENCE ';' .
135i TypeDomain = [ SchemaId '.' ] TypeId .
136i TypeId = type_ref .
137i TypeInstance = TypeInstanceId '=' TypeInstanceValue ';' .
138i TypeInstanceId = simple_id .
139i TypeInstanceValue = TypeDomain '{' TypeValue '}' .

140i TypeValue = AggregationValue | BaseValue | ConstantRef |
                EntityInstanceValue | NamedInstanceValue |
                ObjectInstanceRef | ParameterRef .
141i UseContextBlock = CALL ContextRef ';'
                      UseContextBody END_CALL ';' .
142i UseContextBody = [ ImportSpec ] [ ParameterSpec ] .
```

Примечание – Следующие грамматические правила EXPRESS 156–318, за исключением правил 228, 246, 267, 270, 274, 277–281, 302 и 313, используются в EXPRESS-I.

```
156 abstract_supertype_declaration = ABSTRACT SUPERTYPE [ subtype_constraint ] .
157 actual_parameter_list = '(' parameter { ',' parameter } ')' .
158 add_like_op = '+' | '-' | OR | XOR .
159 aggregate_initializer = '[' [ element { ',' element } ] ']' .

160 aggregate_source = simple_expression .
161 aggregate_type = AGGREGATE [ ':' type_label ] OF parameter_type .
162 aggregation_types = array_type | bag_type | list_type | set_type .
163 algorithm_head = { declaration } [ constant_decl ] [ local_decl ] .
164 alias_stmt = ALIAS variable_id FOR general_ref { qualifier } ';' stmt { stmt }
                 END_ALIAS ';' .
165 array_type = ARRAY bound_spec OF [ OPTIONAL ] [ UNIQUE ] base_type .
166 assignment_stmt = general_ref { qualifier } ':=' expression ';' .
167 attribute_decl = attribute_id | qualified_attribute .
168 attribute_id = simple_id .
169 attribute_qualifier = '.' attribute_ref .

170 bag_type = BAG [ bound_spec ] OF base_type .
171 base_type = aggregation_types | simple_types | named_types .
172 binary_type = BINARY [ width_spec ] .
173 boolean_type = BOOLEAN .
174 bound_1 = numeric_expression .
175 bound_2 = numeric_expression .
176 bound_spec = '[' bound_1 ':' bound_2 ']' .
177 built_in_constant = CONST_E | PI | SELF | ?' .
178 built_in_function = ABS | ACOS | ASIN | ATAN | BLENGTH | COS | EXISTS |
                      EXP | FORMAT | HIBOUND | HIINDEX | LENGTH |
                      LOBOUND | LOINDEX | LOG | LOG2 | LOG10 | NVL |
                      ODD | ROLESOF | SIN | SIZEOF | SQRT | TAN | TYPEOF |
                      USEDIN | VALUE | VALUE_IN | VALUE_UNIQUE .
179 built_in_procedure = INSERT | REMOVE .

180 case_action = case_label { ',' case_label } ':' stmt .
181 case_label = expression .
182 case_stmt = CASE selector OF { case_action } [ OTHERWISE ':' stmt ]
                END_CASE ';' .
183 compound_stmt = BEGIN stmt { stmt } END ';' .
184 constant_body = constant_id ':' base_type ':=' expression ';' .
185 constant_decl = CONSTANT constant_body { constant_body }
                    END_CONSTANT ';' .
```

```
186 constant_factor = built_in_constant | constant_ref .
187 constant_id = simple_id .
188 constructed_types = enumeration_type | select_type .
189 declaration = entity_decl | function_decl | procedure_decl | type_decl .

190 derived_attr = attribute_decl ':' base_type ':=' expression ';'.
191 derive_clause = DERIVE derived_attr { derived_attr } .
192 domain_rule = [ label ':' ] logical_expression .
193 element = expression [ ':' repetition ] .
194 entity_body = { explicit_attr } [ derive_clause ] [ inverse_clause ]
                  [ unique_clause ] [ where_clause ] .
195 entity_constructor = entity_ref '(' [ expression { ',' expression } ] ')' .
196 entity_decl = entity_head  entity_body END_ENTITY ';' .
197 entity_head = ENTITY entity_id [ subsuper ] ';' .
198 entity_id = simple_id .
199 enumeration_id = simple_id .

200 enumeration_reference = [ type_ref '.' ] enumeration_ref .
201 enumeration_type = ENUMERATION OF '(' enumeration_id { ',' enumeration_id } ')' .
202 escape_stmt = ESCAPE ';' .
203 explicit_attr = attribute_decl { ',' attribute_decl } ':' [ OPTIONAL ]
                  base_type ';' .
204 expression = simple_expression [ rel_op_extended  simple_expression ] .
205 factor = simple_factor [ '**' simple_factor ] .
206 formal_parameter = parameter_id { ',' parameter_id } ':' parameter_type .
207 function_call = ( built_in_function | function_ref ) [ actual_parameter_list ] .
208 function_decl = function_head [ algorithm_head ] stmt { stmt }
                  END_FUNCTION ';'  .
209 function_head = FUNCTION function_id [ '(' formal_parameter
                  { ';' formal_parameter } ')' ] ':' parameter_type ';' .

210 function_id = simple_id .
211 generalized_types = aggregate_type | general_aggregation_types| generic_type .
212 general_aggregation_types = general_array_type | general_bag_type |
                                general_list_type | general_set_type .
213 general_array_type = ARRAY [ bound_spec ] OF [ OPTIONAL ] [ UNIQUE ]
                         parameter_type .
214 general_bag_type = BAG [ bound_spec ] OF parameter_type .
215 general_list_type = LIST [ bound_spec ] OF [ UNIQUE ] parameter_type .
216 general_ref = parameter_ref | variable_ref .
217 general_set_type = SET [ bound_spec ] OF parameter_type .
218 generic_type = GENERIC [ ':' type_label ] .
219 group_qualifier = '\' entity_ref .

220 if_stmt = IF logical_expression THEN stmt { stmt } [ ELSE stmt { stmt } ]
          END_IF ';' .
221 increment = numeric_expression .
222 increment_control = variable_id ':=' bound_1 TO bound_2 [ BY increment ] .
223 index = numeric_expression .
224 index_1 = index .
225 index_2 = index .
226 index_qualifier = '[' index_1 [ ':' index_2 ] ']' .
227 integer_type = INTEGER .
<  228 interface_specification = reference_clause | use_clause .  >
229 interval = '{' interval_low  interval_op  interval_item  interval_op
           interval_high '}' .

230 interval_high = simple_expression .
231 interval_item = simple_expression .
232 interval_low = simple_expression .
233 interval_op = '<' | '<=' .
234 inverse_attr = attribute_decl ':' [ ( SET | BAG ) [ bound_spec] OF ] entity_ref
```

```
                    FOR attribute_ref ';' .
235 inverse_clause = INVERSE inverse_attr { inverse_attr } .
236 label = simple_id .
237 list_type = LIST [ bound_spec ] OF [ UNIQUE ] base_type .
238 literal = binary_literal | integer_literal | logical_literal | real_literal |
              string_literal .
239 local_decl = LOCAL local_variable { local_variable } END_LOCAL ';' .

240 local_variable = variable_id { ',' variable_id } ':' parameter_type
                    [ ':=' expression ] ';' .
241 logical_expression = expression .
242 logical_literal = FALSE | TRUE | UNKNOWN .
243 logical_type = LOGICAL .
244 multiplication_like_op = '*' ] '/' | DIV | MOD | AND | '||' .
245 named_types = entity_ref | type_ref .
< 246 named_type_or_rename = named_types [ AS ( entity_id | type_id ) ] . >
247 null_stmt = ';' .
248 number_type = NUMBER .
249 numeric_expression = simple_expression .

250 one_of = ONEOF '(' supertype_expression { ',' supertype_expression } ')' .
251 parameter = expression .
252 parameter_id = simple_id .
253 parameter_type = generalized_types | named_types | simple_types .
254 population = entity_ref .
255 precision_spec = numeric_expression .
256 primary = literal | ( qualifiable_factor { qualifier } ) .'
257 procedure_call_stmt = ( built_in_procedure | procedure_ref )
                          [ actual_parameter_list ] ';' .
258 procedure_decl = procedure_head [ algorithm_head ] { stmt } END_PROCEDURE ';' .
259 procedure_head = PROCEDURE procedure_id [ '(' [ VAR ] formal_parameter
                   { ';' [ VAR ] formal_parameter } ')' ] ';' .

260 procedure_id = simple_id .
261 qualifiable_factor = attribute_ref | constant_factor | function_call |
                         general_ref | population .
262 qualified_attribute = SELF group_qualifier  attribute_qualifier .
263 qualifier = attribute_qualifier | group_qualifier | index_qualifier .
264 query_expression = QUERY '(' variable_id '<*' aggregate_source '|'
                         logical_expression ')' .
265 real_type = REAL [ '(' precision_spec ')' ] .
266 referenced_attribute = attribute_ref | qualified_attribute .
< 267 reference_clause = REFERENCE FROM schema_ref [ '(' resource_or_rename
                        { ',' resource_or_rename } ')' ] ';' . >
268 rel_op = '<' | '>' | '<=' | '>=' | '<>' | '=' | ':<>:' | ':=:' .
269 rel_op_extended = rel_op | IN | LIKE .

< 270 rename_id =      constant_id | entity_id | function_id | procedure_id | type_id . >
271 repeat_control = [ increment_control ] [ while_control ] [ until_control ] .
272 repeat_stmt = REPEAT repeat_control ';' stmt { stmt } END_REPEAT ';' .
273 repetition = numeric_expression .
< 274 resource_or_rename = resource_ref [ AS rename_id ] . >
275 resource_ref = constant_ref | entity_ref | function_ref | procedure_ref | type_ref .
276 return_stmt = RETURN [ '(' expression ')' ] ';' .
< 277 rule_decl = rule_head [ algorithm_head ] { stmt } where_clause
                END_RULE ';' . >
< 278 rule_head = RULE rule_id FOR '(' entity_ref { ',' entity_ref } ')' ';' . >
< 279 rule_id = simple_id . >

< 280 schema_body = { interface_specification } [ constant_decl ]
                   { declaration | rule_decl } . >
```

```
< 281 schema_decl = SCHEMA schema_id ';' schema_body END_SCHEMA ';' . >
282 schema_id = simple_id .
283 selector = expression .
284 select_type = SELECT '(' named_types { ',' named_types } ')' .
285 set_type = SET [ bound_spec ] OF base_type .
286 sign = '+' | '-' .
287 simple_expression = term { add_like_op  term } .
288 simple_factor = aggregate_initializer | entity_constructor |
                    enumeration_reference | interval | query_expression |
                    ( [ unary_op ] ( '(' expression ')' | primary ) ) .
289 simple_types = binary_type | boolean_type | integer_type | logical_type |
                   number_type | real_type | string_type .

290 skip_stmt = SKIP ';' .
291 stmt = alias_stmt | assignment_stmt | case_stmt | compound_stmt | escape_stmt |
           if_stmt | null_stmt | procedure_call_stmt | repeat_stmt | return_stmt |
           skip_stmt .
292 string_literal = simple_string_literal | encoded_string_literal .
293 string_type = STRING [ width_spec ] .
294 subsuper = [ supertype_constraint ] [ subtype_declaration ] .
295 subtype_constraint = OF '(' supertype_expression ')' .
296 subtype_declaration = SUBTYPE OF '(' entity_ref { ',' entity_ref } ')' .
297 supertype_constraint = abstract_supertype_declaration | supertype_rule .
298 supertype_expression = supertype_factor { ANDOR supertype_factor } .
299 supertype_factor = supertype_term { AND supertype_term } .

300 supertype_rule = SUPERTYPE subtype_constraint .
301 supertype_term = entity_ref | one_of | '(' supertype_expression ')' .
< 302 syntax = schema_decl { schema_decl } . >
303 term = factor { multiplication_like_op  factor } .
304 type_decl = TYPE type_id '=' underlying_type ';' [ where_clause ] END_TYPE ';' .
305 type_id = simple_id .
306 type_label = type_label_id | type_label_ref .
307 type_label_id = simple_id .
308 unary_op = '+' | '-' | NOT .
309 underlying_type = constructed_types | aggregation_types | simple_types |
                      type_ref .

310 unique_clause = UNIQUE unique_rule ';' { unique_rule ';' } .
311 unique_rule = [ label ':' ] referenced_attribute { ',' referenced_attribute } .
312 until_control = UNTIL logical_expression .
< 313 use_clause = USE FROM schema_ref [ '(' named_type_or_rename
                  { ',' named_type_or_rename } ')' ] ';' . >
314 variable_id = simple_id .
315 where_clause = WHERE domain._rule ';' { domain_rule ';' } .
316 while_control = WHILE logical_expression .
317 width = numeric_expression .
318 width_spec = '(' width ')' [ FIXED ] .
```

A.5 Список перекрестных ссылок

Конструкция, указанная слева, используется в конструкциях, указанных справа.

0i	CALL	141i
1i	CRITERIA	131i
2i	END_CALL	141i
3i	END_CRITERIA	131i
4i	END_NOTES	132i
5i	END_OBJECTIVE	97i
6i	END_PARAMETER	83i
7i	END_PURPOSE	133i
8i	END_REALIZATION	130i
9i	END_REFERENCES	134i

10i	END_SCHEMA_DATA	109i
11i	END_TEST_CASE	127i
12i	IMPORT	84i
13i	NOTES	132i
14i	OBJECTIVE	97i
15i	PARAMETERi	83i
16i	PURPOSE	133i
17i	REALIZATION	130i
18i	REFERENCES	134i
19i	SCHEMA_DATA	109i
20i	SUBOF	85i
21i	SUPOF	51i
22i	TEST_CASE	127i
23i	USING	112i
24i	WITH	101i 112i
25i	BinaryValue	123i
26i	Description	131i 132i 133i 134i
27i	EncodedStringValue	124i
28i	EnumerationValue	48i 76i 119i
29i	IntegerValue	94i
30i	Nil	52i 81i 98i 106i
31i	SignedMathConstant	104i
32i	SignedRealLiteral	104i
33i	SimplStringValue	124i
34i	ComplexEntityInstanceRef	37i
35i	ConstantRef	103i 105i 140i
36i	ContextRef	141i
37i	EntityInstanceRef	59i 71i 96i 113i
38i	EnumerationInstanceRef	96i 113i
39i	ParameterRef	45i 105i 140i
40i	SelectInstanceRef	96i 113i
41i	SimpleInstanceRef	96i
42i	SimpleEntityInstanceRef	37i
43i	SupSubRef	65i
44i	TypeInstanceRef	59i 96i 113i
45i	ActualParameter	101i
46i	AggregationValue	55i 59i 64i 103i 105i 140i
47i	Assignment	84i
48i	BaseValue	55i 59i 103i 105i 140i
49i	BequeathesTo	65i
50i	BooleanValue	123i
51i	ComplexEntityInstanceId	69i
52i	ConstantBlock	110i
53i	ConstantId	35i 54i
54i	ConstantSpec	52i
55i	ConstantValue	54i 64i
56i	ContextBlock	78i
57i	ContextBody	56i
58i	ContextId	36i 56i
59i	DerattValue	60i 64i
60i	DerivedAttr	70i
61i	DynamicAggr	46i
62i	DynamicEntityRefList	87i
63i	DynamicList	61i
64i	DynamicMember	63i 81i
65i	DynamicSupSubRefList	49i 85i

66i	EntityDomain	70i	93i
67i	EntityId	66i	
68i	EntityInstance	95i	
69i	EntityInstanceId	68i	
70i	EntityInstanceValue	55i 59i 68i 103i 140i	
71i	EntityRefList	62i	
72i	EnumerationDomain	76i 93i	
73i	EnumerationId	72i	
74i	EnumerationInstance	95i	
75i	EnumerationInstanceId	38i 74i	
76i	EnumerationInstanceValue	59i 74i 93i	
77i	ExplicitAttr	70i	
78i	ExpressSyntax		
79i	FixedAggr	46i	
80i	FixedList	79i	
81i	FixedMember	80i	
82i	FormalParameter	83i	
83i	FormalParameterBlock	57i	
84i	ImportSpec	142i	
85i	InheritsFrom	70i	
86i	InvattValue	87i	
87i	InverseAttr	70i	
88i	LogicalValue	52i 123i	
89i	MathConstant	52i	
90i	ModelBlock	78i	
91i	ModelBody	90i	
92i	ModelId	39i 90i	
93i	NamedInstanceValue	55i 103i 105i 119i 140i	
94i	NumberValue	123i	
95i	ObjectInstance	78i 110i	
96i	ObjectInstanceRef	102i 103i 105i 119i 140i	
97i	ObjectiveBlock	128i	
98i	OptattValue	99i	
99i	OptionalAttr	77i	
100i	ParameterId	39i 82i	
101i	ParameterSpec	142i	
102i	ParmValue	45i 64i	
103i	ParmValueDefault	82i	
104i	RealValue	94i	
105i	RequttValue	64i 98i 106i	
106i	RequiredAttr	77i	
107i	RoleName	60i 87i 99i 106i	
108i	SchemaId	66i 72i 109i 114i 135i	
109i	SchemaInstanceBlock	57i 78i 91i	
110i	SchemaInstanceBody	109i	
111i	SchemaReferences	128i	
112i	SchemaReferenceSpec	57i 111i	
113i	SelectableInstanceRef	47i	
114i	SelectDomain	93i 118i	
115i	SelectId	114i	
116i	SelectInstance	95i	
117i	SelectInstanceId	40i 116i	
118i	SelectInstanceValue	93i 116i	
119i	SelectValue	55i 103i 105i 118i	
120i	SimpleEntityInstanceId	34i 42i 51i 69i	

121i	SimpleInstance	| 95i
122i	SimpleInstanceId	| 41i 121i
123i	SimpleValue	| 48i 121i
124i	StringValue	| 123i
125i	SupSubId	| 43i 51i
126i	SupportAlgorithm	| 57i 128i
127i	TestCaseBlock	| 78i
128i	TestCaseBody	| 127i
129i	TestCaseId	| 127i
130i	TestRealization	| 128i
131i	TestCriteria	| 97i
132i	TestNotes	| 97i
133i	TestPurpose	| 97i
134i	TestReference	| 97i
135i	TypeDomain	| 92i 139i
136i	TypeId	| 135i
137i	TypeInstance	| 95i
138i	TypeInstanceId	| 44i 137i
139i	TypeInstanceValue	| 59i 93i 137i
140i	TypeValue	| 55i 59i 64i 103i 105i 119i 139i
141i	UseContextBlock	| 130i
142i	UseContextBody	| 141i
0	ABS	| 178
1	ABSTRACT	| 156
2	ACOS	| 178
3	AGGREGATE	| 161
4	ALIAS	| 164
5	AND	| 244 299
6	ANDOR	| 298
7	ARRAY	| 165 213
8		
9	ASIN	| 178
10	ATAN	| 178
11	BAG	| 170 214 234
12	BEGIN	| 183
13	BINARY	| 172
14	BLENGTH	| 178
15	BOOLEAN	| 173
16	BY	| 222
17	CASE	| 182
18	CONSTANT	| 185 52i
19	CONST_E	| 177 89i
20	CONTEXT	| 56i
21	COS	| 178
22	DERIVE	| 191
23	DIV	| 244
24	ELSE	| 220
25	END	| 183
26	END_ALIAS	| 164
27	END_CASE	| 182
28	END_CONSTANT	| 185 52i
29	END_CONTEXT	| 56i
30	END_ENTITY	| 196
31	END_FUNCTION	| 208

32	END_IF	220
33	END_LOCAL	239
34	END_MODEL	90i
35	END_PROCEDURE	258
36	END_REPEAT	272
37		
38		
39	END_TYPE	304
40	ENTITY	197
41	ENUMERATION	201
42	ESCAPE	202
43	EXISTS	178
44	EXP	178
45	FALSE	242 50i
46	FIXED	318
47	FOR	164 234 278
48	FORMAT	178
49		
50	FUNCTION	209
51	GENERIC	218
52	HIBOUND	178
53	HIINDEX	178
54	IF	220
55	IN	269
56	INSERT	179
57	INTEGER	227 86i
58	INVERSE	235
59	LENGTH	178
60	LIKE	269
61	LIST	215 237
62	LOBOUND	178
63	LOCAL	239
64	LOG	178
65	LOG10	178
66	LOG2	178
67	LOGICAL	243
68	LOINDEX	178
69	MOD	244
70	MODEL	90i
71	NOT	308
72	NUMBER	248
73	NVL	178
74	OOD	178
75	OF	161 165 170 182 201 213 214 215 217 234 237 285 295 296
76	ONEOF	250
77	OPTIONAL	165 203 213
78	OR	158
79	OTHERWISE	182
80	PI	177 89i
81	PROCEDURE	259
82	QUERY	264
83	REAL	265
84		
85	REMOVE	179
86	REPEAT	272

87	RETURN	| 276
88	ROLESOF	| 178
89		
90		
91	SELECT	| 284
92	SELF	| 177 262
93	SET	| 217 234 285
94	SIN	| 178
95	SIZEOF	| 178
96	SKIP	| 290
97	SQRT	| 178
98	STRING	| 293
99	SUBTYPE	| 296
100	SUPERTYPE	| 156 300
101	TAN	| 178
102	THEN	| 220
103	TO	| 222
104	TRUE	| 242 50i
105	TYPE	| 304
106	TYPEOF	| 178
107	UNIQUE	| 165 213 215 237 310
108	UNKNOWN	| 242
109	UNTIL	| 312
110		
111	USEDIN	| 178
112	VALUE	| 178
113	VALUE_IN	| 178
114	VALUE_UNIQUE	| 178
115	VAR	| 259
116	WHERE	| 315
117	WHILE	| 316
118	XOR	| 158
119	bit	| 136
120	digit	| 121 123 127 130 140
121	digits	| 138 139 125i
122	encoded_character	| 137 27i
123	hex_digit	| 133
124	letter	| 127 130 140
125	lparen_not_star	| 142
126	not_lparen_star	| 142
127	not_paren_star	| 126 131 132
128	not_paren_star_quote_special	| 129 130 134
129	not_paren_star_special	| 127
130	not_quote	| 141 33i
131	not_rparen	| 135
132	not_star	| 125
133	octet	| 122
134	special	|
135	star_not_rparen	| 142
136	binary_literal	| 238 25i
137	encoded_string_literal	| 292
138	initeger_literal	| 238
139	real_literal	| 238
140	simple_id	| 168 187 198 199 210 236 252 260 279 282 305 307 314 28i 58i 75i 92i 100i 117i 120i

		122i	129i	138i
141	simple_string_literal	292		
142	embedded_remark	142	143	
143	remark			
144	tail_remark	143		
145	attribute_ref	169	234	261	266	107i
146	constant_ref	186	275	53i
147	entity_ref	195	219	234	245	254	275	278	296	301	67i
148	enumeration_ref	200		
149	function_ref	207	275	
150	parameter_ref	216		
151	procedure_ref	257	275	
152	schema_ref	108i	112i	
153	type_label_ref	306		
154	type_ref	200	245	275	309	73i	115i	136i
155	variable_ref	216		
156	abstract_supertype_declaration	297		
157	actual_parameter_list	207	257	
158	add_like_op	287		
159	aggregate_initializer	288		
160	aggregate_source	264		
161	aggregate_type	211		
162	aggregation_types	171	309	
163	algorithm_head	208	258	
164	alias_stmt	291		
165	array_type	162		
166	assignment_stmt	291	130i	
167	attribute_decl	190	203	234
168	attribute_id	145	167	
169	attribute_qualifier	262	263	
170	bag_type	162		
171	base_type	165	170	184	190	203	237	285
172	binary_type	289		
173	boolean_type	289		
174	bound_1	176	222	
175	bound_2	176	222	
176	bound_spec	165	170	213	214	215	217	234	237	285
177	built_in_constant	186		
178	built_in_function	207		
179	built_in_procedure	257		
180	case_action	182		
181	case_label	180		
182	case_stmt	291		
183	compound_stmt	291		
184	constant_body	185		
185	constant_decl	163		
186	constant_factor	261		
187	constant_id	146	184	
188	constructed_types	309		
189	declaration	163		
190	derived_attr	191		
191	derive_clause	194		
192	domain_rule	315		
193	element	159		
194	entity_body	196		

195	*entity_constructor*	| 288
196	entity_decl	| 189
197	entity_head	| 196
198	entity_id	| 147 197
199	enumeration_id	| 148 201
200	enumeration_reference	| 288
201	enumeration_type	| 188
202	escape_stmt	| 291
203	explicit_attr	| 194
204	expression	| 166 181 184 190 193 195 240 241 251 276 283 288 102i 103i
205	factor	| 303
206	formal_parameter	| 209 259
207	function_call	| 261
208	function_decl	| 189 126i
209	function_head	| 208
210	function_id	| 149 209
211	generalized_types	| 253
212	general_aggregation_types	| 211
213	general_array_type	| 212
214	general_bag_type	| 212
215	general_list_type	| 212
216	general_ref	| 164 166 261
217	general_set_type	| 212
218	generic_type	| 211
219	*group_qualifier*	| 262 263
220	if_stmt	| 291
221	increment	| 222
222	increment_control	| 271
223	index	| 224 225
224	index_1	| 226
225	index_2	| 226
226	index_qualifier	| 263
227	integer_type	| 289
228		
229	interval	| 288
230	interval_high	| 229
231	interval_item	| 229
232	interval_low	| 229
233	interval_op	| 229
234	inverse_attr	| 235
235	inverse_clause	| 194
236	label	| 192 311
237	list_type	| 162
238	literal	| 256
239	local_decl	| 163 130i
240	local_variable	| 239
241	logical_expression	| 192 220 264 312 316
242	logical_literal	| 238 88i
243	logical_type	| 289
244	multiplication_like_op	| 303
245	named_types	| 171 253 284
246		
247	*null_stmt*	| 291
248	number_type	| 289

249	numeric_expression	| 174 175 221 223 255 273 317
250	one_of	| 301
251	parameter	| 157
252	parameter_id	| 150 206
253	parameter_type	| 161 206 209 213 214 215 217 240 82i
254	population	| 261
255	precision_spec	| 265
256	primary	| 288
257	procedure_call_stmt	| 291
258	procedure_decl	| 189 126i
259	procedure_head	| 258
260	procedure_id	| 151 259
261	qualifiable_factor	| 256
262	qualified_attribute	| 167 266
263	qualifier	| 164 166 256
264	query_expression	| 288
265	real_type	| 289
266	referenced_attribute	| 311
267		
268	rel_op	| 269
269	rel_op_extended	| 204
270		
271	repeat_control	| 272
272	repeat_stmt	| 291
273	repetition	| 193
274		
275	resource_ref	| 112i
276	return_stmt	| 291
277		
278		
279		
280		
281		
282	schema_id	| 152
283	selector	| 182
284	select_type	| 188
285	set_type	| 162
286	sign	| 139 29i 31i 32i
287	simple_expression	| 160 204 230 231 232 249
288	simple_factor	| 205
289	simple_types	| 171 253 309
290	skip_stmt	| 291
291	stmt	| 164 180 182 183 208 220 258 272
292	string_literal	| 238
293	string_type	| 289
294	subsuper	| 197
295	subtype_constraint	| 156 300
296	subtype_declaration	| 294
297	supertype_constraint	| 294
298	supertype_expression	| 250 295 301
299	supertype_factor	| 298
300	supertype_rule	| 297
301	supertype_term	| 299
302		
303	term	| 287

304	type_decl	| 189
305	type_id	| 154 304
306	type_label	| 161 218
307	type_label_id	| 153 306
308	unary_op	| 288
309	underlying_type	| 304
310	unique_clause	| 194
311	unique_rule	| 310
312	until_control	| 271
313		
314	variable_id	| 155 164 222 240 264 47i
315	where_clause	| 194 304
316	while_control	| 271
317	width	| 318
318	width_spec	| 172 293

ПРИЛОЖЕНИЕ В
(обязательное)

Заявка о соответствии реализации протоколу (ЗСРП)

Является ли данная реализация синтаксическим анализатором/верификатором языка EXPRESS? Если да, то должны быть даны ответы на вопросы, приведенные в В.1.

В.1 Синтаксический анализатор языка EXPRESS-I

Для какого из уровней заявляется поддержка:

☐ уровень 1 – проверка ссылок;

☐ уровень 2 – проверка типов;

☐ уровень 3 – проверка значений;

☐ уровень 4 – полная проверка.

(Примечание — Для того чтобы заявить о поддержке конкретного уровня, должна быть обеспечена поддержка всех нижних уровней).

Каково максимальное целочисленное значение [integer_literal]?: .....:
Какова максимальная точность действительных значений [real_literal]?: .....:
Каков максимальный показатель степени действительных значений [real_literal]?: .....:
Какова максимальная длина строки (в символах) [simple_string_literal]?: .....:
Какова максимальная длина строки (в восьмибитовых байтах) [encoded_string_literal]?: .....:
Какова максимальная длина двоичных значений (в битах) [binary_literal]?: .....:
Существует ли ограничение на общее количество объявленных уникальных идентификаторов? Если да, то чему оно равно?: .....:
Существует ли ограничение на количество символов в идентификаторе? Если да, то чему оно равно?: .....:
Существует ли ограничение на глубину вложения областей применения? Если да, то чему оно равно?: .....:
Как представлена стандартная константа '?' [built_in_constant]?: .....:

ПРИЛОЖЕНИЕ С
(обязательное)

Регистрация информационного объекта

Для того чтобы обеспечить однозначную идентификацию информационного объекта в открытой системе, настоящему стандарту присвоен идентификатор объекта:

{ iso standard 10303 part(12) version(1) }

Смысл этого значения определен в соответствии с ИСО/МЭК 8824-1 и уточнен в ГОСТ Р ИСО 10303-1.

ПРИЛОЖЕНИЕ D
(справочное)

Синтаксис спецификации языка

Нотация, используемая для представления синтаксиса языка EXPRESS-I, установлена в ГОСТ Р ИСО 10303-11. *В настоящем приложении она приведена в качестве справочного материала.*

Полный синтаксис языка EXPRESS-I определен в приложении А. Фрагменты этих синтаксических правил воспроизведены в разных разделах настоящего стандарта для иллюстрации синтаксиса конкретных операторов. Эти фрагменты не всегда полны, так что иногда необходимо обращаться к приложению А для просмотра пропущенных правил. Фрагменты синтаксиса в основной части настоящего стандарта представлены *в прямоугольных рамках. Каждое правило внутри синтаксической рамки имеет слева уникальный номер* для использования его в перекрестных ссылках с другими синтаксическими правилами.

D.1 Синтаксис спецификации

Синтаксис EXPRESS (и EXPRESS-I) устанавливается на основе нотации, производной от Синтаксической нотации Вирта (СНВ); см. для справок [3] из приложения Н.

Нотационные обозначения 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 конструкцию языка. Элемент слева определяется как комбинация элементов справа. Любое число пробелов, появляющихся между элементами конструкции, не имеет значения, пока пробелы не появятся внутри литерала. Конструкция заканчивается точкой `.`.

- использование идентификатора внутри фактора обозначается нетерминальным символом, который появляется слева от другой конструкции. Идентификатор образуется из букв, цифр и символа подчеркивания. Ключевые слова языка представляются конструкциями, идентификаторы которых состоят только из заглавных букв;

- слово «литерал» используется для обозначения терминального символа, который не может быть расширен. Литералом является независимая от регистра последовательность символов, заключенная в апострофы. Символ в данном случае представляет собой любой символ, определяемый в ИСО/МЭК 10646-1 ячейками 21-7E в группе 00, проекции 00, строке 00. Для включения в литерал апострофа он должен быть записан дважды;

- семантики охватывающих скобок определены ниже:
  - фигурные скобки `{ }` означают ноль или более повторений;
  - квадратные скобки `[ ]` означают необязательные параметры;
  - круглые скобки `( )` означают, что группа конструкций, заключенная в круглые скобки, должна использоваться как единая конструкция;
  - вертикальная черта `|` означает, что должен быть выбран только один из термов выражения.

Примечания

1 В настоящем стандарте к описанному выше мета-языку добавлена еще одна конструкция: комментарий. Комментарием является любой текст, заключенный в угловые скобки. Например, < Комментарий > - это комментарий.

2 В частности, комментарий < как в EXPRESS > используется для обозначения того, что конструкция определена в ГОСТ Р ИСО 10303-11 и, в целях совместимости между документами, не повторяется в настоящем стандарте.

Пример 69 – Синтаксис для литерала действительного числа имеет вид:

```
  Синтаксис:
190 real_literal = integer_literal `.` [ integer_literal ]
                   [ `e` [ sign ] integer_literal ] .
163 integer_literal = digit { digit } .
```

Полное определение синтаксиса (приложение А) содержит определения для **sign** и **digit**.

Пример 70 – В соответствии с синтаксисом, приведенным в примере 69, возможны следующие альтернативы:

a) 123.
b) 123.456
c) 123.456e7
d) 123.456E-7

D.2 Нотация специального символа

Следующая нотация используется для представления полных наборов символов и некоторых специальных символов, которые сложно отображать:

- \a представляет символы в ячейках 21-7E строки 00, проекции 00, группы 00 из ИСО/МЭК 10646-1;
- \n представляет новую строку (в зависимости от системы);
- \q является символом апострофа (`) и содержится внутри \a;
- \s является символом пробела;
- \o представляет символы в ячейках 00-1F и 7F строки 00, проекции 00, группы 00 из ИСО/МЭК 10646-1.

ПРИЛОЖЕНИЕ Е
(справочное)

Некоторые контрольные примеры

В настоящем приложении приведены некоторые абстрактные контрольные примеры. Эти примеры не претендуют на роль обязательных абстрактных контрольных примеров, задаваемых в других стандартах серии ГОСТ Р ИСО 10303, и приведены исключительно в иллюстрационных целях.

Начнем с простой EXPRESS-схемы (SCHEMA), для которой определяется контрольный пример.

```
*)
SCHEMA people;

TYPE name = STRING; END_TYPE;

ENTITY person;
   named : name;
   children : SET [0:?] OF person;
END_ENTITY;

ENTITY male
   SUBTYPE OF (person);
END_ENTITY;

ENTITY female;
   SUBTYPE OF (person);
END_ENTITY;

ENTITY married;
   husband: male;
   wife          : female;
END_ENTITY;

END_SCHEMA;
(*
```

E.1 Контрольный пример 1

Этот контрольный пример устанавливает, что должны быть созданы три экземпляра объекта **person**.

```
*)
TEST_CASE test_cube_1;

   WITH people USING(person);

   OBJECTIVE
      PURPOSE To test the creating of supertypes with no subtypes. END_PURPOSE;
      REFERENCES None. END_REFERENCES;
      CRITERIA Three instances of children PERSON shall be created.
      END_CRITERIA;
      NOTES None. END_NOTES;
   END_OBJECTIVE;

   REALIZATION

      LOCAL                                  - - определяем переменные типа person
         p1 : person;
         p2 : person;
         p3 : person;
      END_LOCAL;

      p1 := person(`Alpha`, [ ]);- - создаем экземпляры person
      p2 := person(`Beta`, [ ]);
      p3 := person(`Gamma`, [ ]);

   END_REALIZATION;

END_TEST_CASE;
(*
```

Одним из возможных фрагментов результирующих данных этого контрольного примера является:

```
*)
MODEL case_1;
   SCHEMA_DATA people;
   n1 = name {`Alpha`};
   n2 = name {`Beta`};
   n3 = name {`Gamma`};
   p1 = person{named      -> @n1;
               children   -> ( );}
   p2 = person{named      -> @n2;
               children   -> ( );}
   p3 = person{named      -> @n3;
               children   -> ( );}
   END_SCHEMA_DATA;
END_MODEL;
(*
```

Для последующего использования определяется следующий контекст, основанный на контрольном примере:

```
*)
CONTEXT context_1;
   SCHEMA_DATA people;
   p1[1] = person{named    -> `Alpha`;
                  children -> ( );
                  SUPOF( );};
   p2[1] = person{named    -> `Beta`;
                  children -> ( );
                  SUPOF( );};
   p3[1] = person{named    -> `Gamma`;
                  children -> ( );
                  SUPOF( );};
   END_SCHEMA_DATA;
END_CONTEXT;
(*
```

E.2 Контрольный пример 2

Данный контрольный пример создает подтипы **male** и **female** объекта **person**.

```
*)
TEST_CASE test_case_2;
   WITH people USING(male, female);
   OBJECTIVE To test the creation of subtypes. END_PURPOSE;
      CRITERIA One instance of childless MALE and one of a childless
                  FEMAIL shall be created. END_CRITERIA;
   END_OBJECTIVE;
   REALIZATION
      LOCAL                  - - определяем переменные требуемых типов
         m1  : male;
         f1          female;
      END_LOCAL;
      m1  := person(`Adam`, [ ]) || male( );    - - создаем экземпляр male
      f1          := person(`Eve`, [ ]) || female( );    - - создаем экземпляр female
   END_REALIZATION;
END_TEST_CASE;
(*
```

Одним из возможных фрагментов результирующих данных этого контрольного примера является:

```
*)
MODEL case_2;
   SCHEMA_DATA people;

      m1[1] = person{named      -> `Adam`;
                     children -> ( );
                     SUPOF( @2);};

      m2[2] = male{ SUBOF(@1);};

      f1[1] = person{ named      ->`Eve`;
                     children -> ( );
                     SUPOF( @2);};

      f1[2] = female{ SUBOF(@1);};

   END_SCHEMA_DATA;
END_MODEL;
(*
      Для последующего использования создается также следующий параметризованный контекст.

*)
CONTEXT context_2;

   WITH people USING(person);

   PARAMETER
      c1 : SET OF person := ( );        - - параметр по умолчанию – пустое множество
      c2 : SET OF person := ( );
   END_PARAMETER;

   SCHEMA_DATA people;

   p4[1] = person{named   -> `Adam`;
                  children -> c1;              - - параметризуется атрибут children
                  SUPOF(@2);};

   p4[2] = male{SUBOF(@1);};

   p5[1] = person{named   -> `Eve`;
                  children -> c2;
                  SUPOF( @2);};

   p5[2] = female{SUBOF(@1);};

   END_SCHEMA_DATA;
END_CONTEXT;
(*
      E.3 Контрольный пример 3
      Этот пример создает экземпляр объекта married.

*)
TEST_CASE test_case_3;

   WITH people USING (married);

   OBJECTIVE
      PURPOSE To test the creation of an entity with attributes of type entity.
      END_PURPOSE;
      CRITERIA One instance of a MARRIED entity shall be created.
      END_CRITERIA;
   END_OBJECTIVE;

   REALIZATION

      LOCAL                          - - определяем переменные требуемых типов
         reg: married;
         h1 : male;
         w1 : female;
```

```
    END_LOCAL;
    CALL context_2                  - - используем данные из CONTEXT context_2
        IMPORT(h1     := @p4;
               w1     := @p5; );
    END_CALL;
    reg := married(h1, w1);         - - создаем экземпляр married
  END_REALIZATION;
END_TEST_CASE;
(*
```

Одним из возможных фрагментов результирующих данных этого контрольного примера является:

```
*)
MODEL case_3;
  SCHEMA_DATA people;
  h1[1] = person{named    -> `Adam`;
                 children -> ( );
                 SUPOF(@6);};
  h1[6] = male{SUBOF(@3);};
  w1[7] = person{named    -> `Eve`;
                 children -> ( );
                 SUPOF(@8);};
  w1[8] = female{ SUBOF(@7);};
  reg = married{husband          -> @h1;
                wife             -> @w1;};
  END_SCHEMA_DATA;
END_MODEL;
(*
```

E.4 Контрольный пример 4

Этот контрольный пример собирает множество уже существующих параметризованных данных, а также создает новые данные.

```
*)
TEST_CASE test_case_4;
  WITH people USING(person, male, female, married);
  OBJECTIVE
    PURPOSE To test the creation of married couple with children. END_PURPOSE;
    CRITERIA Three instances of PERSON shall be created. One instance each
             of MALE and FEMALE with children shall be created. One
             instance of MARRIED entity shall be created. END_CRITERIA;
  END_OBJECTIVE;
  REALIZATION
    LOCAL                       - - определяем переменные требуемых типов
      p1 : person;
      p2 : person;
      p3 : person;
      m1 : male;
      f1 : female;
      reg : married;
  END_LOCAL;
  CALL context_1
    IMPORT(p1 := @p1;           - - используем данные из CONTEXT context_1
           p2 := @p2;
           p3 := @p3;);
  END_CALL;
```

```
      CALL context_2;
         IMPORT(m1 := @p4;                    - - используем данные из CONTEXT context_2
                f1 := @p5;
         WITH(c1 := [p1, p3];                 - - множество значений параметра
              c2 := [p2, p3];);
      END_CALL;

      reg := married(m1, f1);                 - - создаем экземпляр married

   END_REALIZATION;

END_TEST_CASE;
(*
```

Одним из возможных фрагментов результирующих данных этого контрольного примера является:

```
*)
MODEL case_4;
   SCHEMA_DATA people;

   n1 = name{`Alpha`};
   n2 = name{`Beta`};
   n3 = name{`Gamma`};

   p1 = person{named      -> @n1;
               children -> ( );}

   p2 = person{named      -> @n2;
               children -> ( );}

   p3 = person{named      -> @n3;
               children -> ( );}

   m1[1] = person{named -> `Adam`;
                  children -> (@p1, @p3);
                  SUPOF(@2);};

   m1[2] = male{SUBOF(@1);};

   f1[1] = person{named   -> `Eve`;
                  children -> (@p2, @p3);
                  SUPOF(@2);};

   f1[2] = female{SUBOF(@1);};

   reg = married{husband -> @m1;
                 wife    -> @f1;};

   END_SCHEMA_DATA;
END_MODEL;
(*
```

ПРИЛОЖЕНИЕ F
(справочное)

Замечания по применению стандарта

В настоящем приложении рассматриваются некоторые потенциальные сферы применения языка EXPRESS-I.

В предметно-ориентированной терминологии EXPRESS-объект (*entity*) следовало бы назвать классом (*a class*), а экземпляр класса — предметом (*an object*), один предмет может ссылаться на другой предмет. В языке EXPRESS различают объекты (*entities*) и типы (*types*) (то есть ENUMERATION, SELECT и другие определяемые типы данных) тем, что объекты могут иметь подтипы, тогда как типы не могут иметь подтипов. Физический файл, определяемый по ГОСТ Р ИСО 10303-21, четко различает объекты и типы тем, что только экземпляры объектов могут появляться в файле, а значения типов встраиваются в значения атрибутов и на них нельзя ссылаться. В языке EXPRESS-I экземпляры объектов трактуются как предметы в предметно-ориентированном смысле. Также допускается трактовка типов как предметов в смысле наличия их экземпляров, на которые можно ссылаться; альтернативно допускается трактовка типов как физического файла, в котором содержатся значения типов.

F.1 Примеры EXPRESS-данных

Простейшим применением языка EXPRESS-I являются упражнения по написанию на бумаге примеров данных, определяющих конструкции языка EXPRESS. Язык позволяет отобразить экземпляры предметов в виде предметов, на которые допускаются ссылки. Экземпляры типов также могут отображаться как предметы, доступные для ссылок, или они могут появляться в значениях других предметов как значения недоступные для ссылок. Примеры, приведенные в настоящем стандарте, показывают обе формы наполнения типов.

Также требуются значения явных атрибутов объектов. Нет необходимости отображать значения вычисляемых или инверсных атрибутов, за исключением оговоренных в примерах, потому что эти значения необходимо вычислять из значений явных атрибутов.

Примеры EXPRESS-схем можно отображать так же, как и отдельные предметы.

EXPRESS-I-конструкция MODEL предназначена для отображения нескольких схем. Обычно конструкция MODEL используется, когда две или более EXPRESS-схемы взаимодействуют друг с другом. Отметим, что сам язык EXPRESS не поддерживает данную конструкцию.

F.2 Абстрактные контрольные примеры

EXPRESS-I-конструкция TEST_CASE предусмотрена для формального определения контрольных примеров, проверяющих реализацию конструкций, установленных в языке EXPRESS. В самом языке EXPRESS эквивалентная конструкция отсутствует.

Для контрольного примера должен быть определен базовый набор предметов EXPRESS-I, состоящий из подлежащих тестированию предметов и относящихся к ним данных. Значения этих предметов могут быть представлены в виде параметров, формальные определения которых заданы в обобщающем контексте (CONTEXT). Затем ряд контрольных примеров может быть определен на основе CONTEXT путем задания фактических значений параметров. Тем самым единый «параметризованный» контекст может поддерживать много различных тестов (испытаний). Так же должна быть представлена документация по контрольному примеру, охватывающая назначение теста и ожидаемые результаты (см. стандарты серии ГОСТ Р ИСО 10303 по аттестационному тестированию).

F.3 Объектные базы

Предполагается наличие некоторой объектной базы, хранящей предметы, соответствующие определяемым EXPRESS-схемам. Это значит, что объектная база имеет возможность обслуживания конкретных предметов, соответствующих EXPRESS-схемам, в которых объявлены их определения. Проектирование и реализация такой объектной базы предлагается читателю в качестве упражнения.

F.3.1 Вход

В заданной объектной базе EXPRESS-I может быть использован как средство ввода предметов в объектную базу. Этот процесс мог быть либо пакетным, когда заранее подготовленный файл EXPRESS-I читается объектным процессором, либо интерактивным, когда пользователь постепенно добавляет предметы EXPRESS-I.

В зависимости от развитости объектной базы пользователю может или не может потребоваться явное задание значений вычисляемых и инверсных атрибутов.

F.3.2 Выход

В заданной наполненной объектной базе EXPRESS-I может быть использован как язык вывода данных для отображения части или всего содержимого объектной базы, воспринимаемого человеком.

В зависимости от развитости объектной базы, отображаемые объектные предметы могут или не могут включать значения вычисляемых и инверсных атрибутов. Однако, по меньшей мере имена ролей этих атрибутов выводить необходимо.

EXPRESS-I-конструкция MODEL спроектирована для отображения совокупности объектной базы.

F.3.3 Тестирование программы (кода)

В идеале реализация объектной базы должна обеспечивать функциональные возможности для оценки всех ограничений на объекты и типы EXPRESS, которые могут быть представлены предметами или значениями в объектной базе. Например EXPRESS-схема может содержать определение объекта (ENTITY), включающее вычисляемый атрибут и ограничение на вычисляемое значение. Объектная база должна обладать возможностями как определения вычисляемого атрибута, так и исключения любого предмета данного класса ENTITY, значения которого не удовлетворяют ограничениям. Для этого требуется программа на каком-либо языке программирования. EXPRESS-I может использоваться при вводе данных для тестирования такой программы.

Другими примерами программ (кодов) являются:

- определение значений инверсных атрибутов;
- проверка уникальности ограничений на совокупность предметов;
- программа реализации определяемых в EXPRESS правил (RULE).

Заметим, что эти типы функций необходимы также для систем тестирования физического файла и других видов процессоров обмена данными.

F.4 Примеры данных, отличных от EXPRESS

Поскольку экземпляры объектов EXPRESS-I имеют форму поименованных кортежей, их можно использовать также для отображения предметов или записей из языков, отличных от EXPRESS. Например экземпляры Си-структур или состояний предметов, представляющие собой экземпляры классов объектно-ориентированных языков типа Си++ или Эйфель, могут быть отображены при помощи EXPRESS-I. Аналогичным образом, EXPRESS-I можно использовать в качестве механизма отображения в языках, поддерживающих фреймы.

Пример 71 — Структура на языке Си может быть определена следующим образом:

```
struct point {
        int x;
        int y;
};
```

Экземпляр на языке EXPRESS-I для этой структуры мог быть представлен в виде:

```
p1 = point {x -> 10;
        y -> 20;};
```

Язык можно использовать для представления табличных данных из реляционных баз данных, где имя объекта эквивалентно имени таблицы, а каждый экземпляр является (идентифицируемой) строкой в таблице, либо сети в объектно-ориентированных баз данных. В другом случае язык может быть использован в качестве файла для данных IGES (международного стандартного обмена графическими данными), независимого от формата представления.

Пример 72 — Таблица реляционной базы данных может быть определена в SQL следующим образом:

```
CREATE TABLE PART
    ( ID          CHAR(6)          NOT  NULL;
      PNAME       CHAR(20)         NOT  NULL;
      COLOR       CHAR(6)          NOT  NULL;
      WEIGHT      SMALLINT         NOT  NULL;
      CITY CHAR(15)                NOT  NULL;
    PRIMARY KEY ( ID ) ;
```

Экземпляры двух строк наполнения таблицы PART можно представить в EXPRESS-I следующим образом:

```
part_row1 = PART{ID -> `p33`;
                 PNAME -> `Nut`;
                 COLOR -> `Red`;
                 WEIGHT -> 12;
                 CITY -> `Paris;  };
part_row2 = PART{ID -> `p8`;
                 PNAME -> `Washer`;
                 COLOR -> `Green`;
                 WEIGHT -> 4;
                 CITY -> `Rome`; };
```

Пример совершенно иного использования дан Гудвином [4], который предложил EXPRESS-I в качестве формального мета-языка для Семантически Унифицированной Мета Модели [5], базирующейся, в свою очередь, на логике предикатов.

ПРИЛОЖЕНИЕ G
(справочное)

Технические подходы

В данном приложении описаны некоторые технические подходы, использованные при регламентации требований к языку EXPRESS-I, описанному в настоящем стандарте. Настоящий материал содержит хронологическое и тематическое описание дискуссий по вопросам регламентации требований к языку EXPRESS-I и результаты принятых по данным вопросам решений.

Язык EXPRESS-I был разработан в начале 1990 г. с целью удовлетворения потребностей пользователей в написании простых примеров программ EXPRESS-моделей, применяемых для проверки и понимания моделей. В связи с этим описание языка ограничивалось только отображением экземпляров объектов. Первые версии документа планировались как дополнение к справочному руководству по языку EXPRESS (ГОСТ Р ИСО 10303-11). Позднее описание языка было существенно расширено.

G.1 Абстрактные контрольные примеры

Сан-Диего, апрель 1991 г.: Язык EXPRESS-I соответствует своему назначению, но можно ли его расширить для работы с контрольными примерами, например для определения параметризованных экземпляров?

Обсуждение/Решение: Следующая версия будет расширена в соответствии с данным предложением. Кроме того, хотя физический файл не позволяет включать независимые экземпляры типов (TYPE), было бы желательно их включение в EXPRESS-I, чтобы другие виды реализации стандартов серии ГОСТ Р ИСО 10303 (ИСО 10303) могли их рассматривать в качестве предметов первого класса.

G.2 Связь с EXPRESS

Саппоро, июль 1991 г.: Насколько тесной должна быть связь между EXPRESS-I и EXPRESS? Теперь, при обеспечении описания контрольных примеров, следует ли считать EXPRESS-I более ориентированным на класс контрольного примера, чем на класс методов описания?

Обсуждение/Решение: Язык EXPRESS-I очевидно нуждается в тесной корреляции с существующей лексической языка EXPRESS, а также с его возможными расширениями в EXPRESS версии 2. Возможно, его следует сохранить как класс документа по методам описания (метод языка для описания). Однако, чтобы подчеркнуть разницу между описаниями информационной модели (например, EXPRESS) и описаниями реализаций и (или) тестирования, описание EXPRESS-I следует выделить в отдельный документ, а не выпускать в виде приложения к EXPRESS. РГ3/П3 обратилась в Саппоро к Секретариату ТК 184 (PMAG) с просьбой выделить описание EXPRESS-I в отдельный документ. До публикации в виде документа с конкретным обозначением его следует переписывать в виде отдельного документа, а не в виде приложения.

G.3 Ссылки на предметы

Саппоро, июль 1991 г.: Почему имеется знак "@" перед ссылками на объект или тип?

Обсуждение/Решение: Главным пожеланием при разработке языка являлось установление лексического различия между областями значений. Это подразумевает, что лексическое представление значения должно, насколько возможно, указывать соответствующую область значения. Поэтому знак "@" используется для установления отличия того, что в языках программирования называется указателями, от других элементов значений, например целочисленных или переменных.

G.4 Агрегации

Саппоро, июль 1991 г.: Нужно ли лексически обозначать область значения каждого вида агрегации? То есть следует лексически различать мультимножества (bags), списки (lists) и наборы (sets), так как все они отличаются от массивов (arrays).

Обсуждение/Решение: Возможно, но это усложнило бы язык. На данный момент создан язык, описывающий различия между агрегациями фиксированной и переменной длины в качестве первичной характеристики поведения. Внутреннее поведение (то есть упорядочение и копирование) менее важно. В любом случае, существует базовое допущение о том, что все области значений устанавливаются вне EXPRESS-I.

Саппоро, июль 1991 г.: Нужно ли в языковых конструкциях иметь возможность установления максимального числа символов в строке, границ массива и т.д.?

Обсуждение/Решение: Нет. Имеется базовое допущение о существовании концептуальной модели (не обязательно описанной на языке EXPRESS), которая определяет эти характеристики. Язык EXPRESS-I используется для отображения совокупности примеров концептуальной модели.

G.5 Строковые значения

Саппоро, июль 1991 г.: Следует ли рассматривать возможность включения новых строк в строковые значения в противоположность языку EXPRESS?

Обсуждение/Решение: EXPRESS можно рассматривать как язык (концептуального) определения, и в этом смысле строка может быть неограниченной длины. EXPRESS-I ближе к языку реализации по крайней мере в смысле способности отображать строковые и другие значения. Материал (например, бумага, экран монитора), на котором отображаются данные, ограничен в размерах. Поэтому обеспечивается механизм разбиения длинных строк на более короткие с целью их отображения.

G.6 Тестирование и принятие модели

Саппоро, июль 1991 г.: Хотя данная версия EXPRESS-I и поддерживает определение абстрактного контрольного примера, достаточно ли этих средств?

Обсуждение/Решение: Мы не знаем. Исходные данные и требования к тестированию активно изучаются, например руководителем РГ6.

G.7 Расширение возможностей контрольного примера

В период с июля 1991 г. по июнь 1992 г. были получены три документа, отражающие позиции членов РГ6 в отношении требований к возможностям контрольных примеров:

- Mark Davies, Requirements for an Instantiation Language for EXPRESS, CADDETC Document D/91/0037, 9 October 1991;
- Mark Davies, EXPRESS-I Requirements, TC184/SC4/WG5/P3 N??, 22 January 1992;
- Paul Bell, Enhancements needed to EXPRESS-I to support ATC development, CADDETC Document CTS2/92/L/001/t, 1 June 1992.

Последний из этих отчетов фактически включал содержимое предыдущих и был гораздо более существенным документом.

Язык EXPRESS-I был модифицирован в июне 1992 г. с учетом требований, предложенных в этих отчетах. Это привело к коренной переработке документа.

G.8 Соответствие языку EXPRESS

На встрече в Лондоне (июль 1992 г.) был рассмотрен документ от июня 1992 г. и согласованы незначительные технические изменения в нем.

Одновременно, поскольку документ корректировался с целью включения этих изменений, он был редакционно и технически структурирован в соответствии с проектом стандарта на язык EXPRESS. В результате в него были внесены основные изменения в части введения набора символов по ИСО/МЭК 10646-1.

G.9 Опытная апробация

На встрече в Далласе в 1992 г. РГ6 решила разработать опытные абстрактные контрольные примеры на основе версии EXPRESS-I от ноября 1992 г. Эти эксперименты отчасти предназначались для определения достаточности требований к языку описания абстрактных контрольных примеров (АТС), а также для выявления дополнительных требований и, при наличии таковых, - документирования этих требований.

В это время РГ6 уже располагала некоторыми дополнительно предложенными требованиями, но было решено не включать их в язык (за исключением представления их в форме замечаний в комментариях к опытным тестам), пока работа по опытным контрольным примерам не будет рассмотрена в начале 1993 г.

Затем было отмечено, что не определено отображение переобъявляемых атрибутов. Кроме того, было бы полезным ввести в контрольный пример более четкое различие между данными управления и тестирования. Было решено добавить недостающее отображение и ввести конструкцию REALIZATION. Без учета этих изменений, описание языка должно было окончательно оформиться к началу 1993 г.

G.10 Расширения алфавита

Даллас, октябрь 1992 г.: В версии 2 EXPRESS имеется требование по поддержке неанглийских алфавитов в комментариях и идентификаторах. Это требование также применимо к EXPRESS-I.

Обсуждение/Решение: Конкретных действий намечено не было. Включение данных требований будет рассматриваться и возможное решение может быть включено в следующую (1993 г.) версию требований к языку.

G.11 Отображение супертипов

Iaian Morison, декабрь 1992 г.: В EXPRESS-I каждый EXPRESS-объект наполняется как сложный экземпляр с отдельным идентификатором и отношениями супертип—подтип, объявляемыми посредством “указателей” SUPOF и SUBOF. Это приводит к поддержке двух неверных представлений:

a) что один экземпляр имеет несколько возможных идентификаторов;

b) не исключается возможность того, что одно и то же множество значений супертипа совместно используется несколькими экземплярами подтипа.

Предлагается представлять сложный экземпляр аналогично определяемым наборам в EXPRESS:

```
i1 = me&sibling {
                g_name --> 'Gr an';
                p_name --> 'Dad';
                my_name --> 'self';
                s_name --> 'Sis'; };
```

Обсуждение/Решение: Похоже имеются три основных варианта отображения экземпляров супертипа:

a) идентифицировать лист и наследовать все атрибуты — проблема в том, что из-за конструкций ANDOR возможна множественность листьев;

b) идентифицировать корень (самый верхний супертип) и двигаться в направлении атрибутов потомков — проблема в том, что возможна множественность корней из-за множественного наследования;

c) трактовать все компоненты одинаково.

Был выбран третий вариант, поскольку он подразумевает, что не требуется оперировать со специальными случаями. Как вы заметили, снизу это означает, что единственный сложный экземпляр может иметь

несколько возможных идентификаторов. Фактически все они являются переименованиями друг друга и могут быть выявлены прохождением по ссылкам SUPOF и SUBOF в EXPRESS-I.

Если смотреть на эту схему сверху, то экземпляр атрибута некоторого другого объекта может ссылаться на соответствующий тип экземпляра объекта в комплексе супертипа. Например, атрибут типа **sibling** может ссылаться на экземпляр **sibling** (и он получает при этом все другие ссылки в комплексе).

Экземпляр, образующий часть одного комплекса супертипа, не может образовывать часть экземпляра другого комплекса супертипа. Это должно быть четко записано в руководстве по языку.

G.12 Комментарии по голосованию за CD–1995

Описание языка EXPRESS-I было предложено для голосования в качестве первой редакции проекта стандарта (CD) в 1995 г. Из-за большого разнообразия замечаний, явившихся результатом голосования, было решено издавать EXPRESS-I как технический отчет, а не как стандарт. Основным пунктом расхождения во мнениях голосовавших был раздел языка "абстрактный контрольный пример": некоторым он понравился, для других он был неприемлем.

Документ вида ТО (технический отчет) включает много редакционных изменений, предложенных при голосовании за CD, и других редакционных изменений, уточняющих понятия. Было сделано одно дополнительное техническое изменение, а в целом документ остался таким, каким он рассылался для голосования. Далее приводятся основные технические замечания, полученные при голосовании.

G.12.1 Обеспечение контрольного примера

Великобритания одобрила материал и использует его;

Франция имела некоторые технические замечания.

Некоторые представители США хотели исключить из языка эту часть, тогда как другие ощущали ее недостаточную проработанность.

За исключением EXPRESS-I, в серии ИСО 10303 (ГОСТ Р ИСО 10303) нет формального языка для абстрактного контрольного примера. Однако, поскольку РГ6 пока не имеет полного множества требований к такому языку, в частности, относительно тестирования реализаций, базирующихся на SDAI, возможно преждевременно стандартизовать данную часть EXPRESS-I. В основном, это заключение и привело к тому, чтобы издавать описание языка в качестве ТО, а не стандарта.

G.12.2 Сложные экземпляры объектов

Повторяющейся темой в комментариях ряда стран при голосовании было неприятие метода наполнения сложных экземпляров объекта (то есть когда экземпляр является иерархией наследования). Консенсусом голосования было решено, что у экземпляра должен быть единственный идентификатор. Это замечание было принято в языке, который, как описывается теперь в настоящем стандарте, определяет единственный идентификатор для сложного экземпляра.

G.12.3 Экземпляры типа

Швейцария возражала против возможности идентификации экземпляров EXPRESS-конструкций, отличных от объектов.

EXPRESS исходит из предположения, что каждый экземпляр объекта (в объектной базе) будет иметь уникальный идентификатор. В объектно-ориентированной терминологии он называется Oid–Object Identifier. Однако EXPRESS ничего не говорит относительно идентификаторов (Oids) для необъектов: ни запрещает их, ни допускает их существования.

Одной из целей проектирования EXPRESS-I было обеспечение возможности демонстрации экземпляров так, как они могли бы быть представлены в некоторой объектной базе (которую EXPRESS называет реализацией). EXPRESS не определяет среду реализации. Поэтому разработчик может выбрать как хранить экземпляры данных, давать уникальные Oids для поддержки экземпляров объектов. EXPRESS-I преднамеренно распространяет понятие Oid на другие виды экземпляров. Язык Smalltalk делает тоже самое, рассматривая все как предмет. Разумеется, не требуется, чтобы возможности идентифицируемых экземпляров необъектов использовались, но они в языке имеются на случай необходимости.

ПРИЛОЖЕНИЕ Н
(справочное)

Библиография

[1] ИСО/ТО 9007—87*) Системы обработки информации. Концепции и терминология для концептуальной схемы и информационной базы

[2] ИСО/МЭК 6429—92*) Информационная технология. Управляющие функции для кодированных наборов символов

[3] WIRTH, H.; "What can we do about the unnecessary diversity of notation for syntactic definitions?", Communications of the ACM, November 1977, vol 20, no. 11, p 822.

[4] GODWIN, A. N., GIANNASI, F. And TAHZIB, S.; "An example using the SUMM with EXPRESS and relational models", in WILSON, P. R. (editor) EUG'94: 4th Annual EXPRESS User Group International Conference, Greenville, SC, 13—14 October, 1994

[5] FULTON, J. A. et al; "Technical report on the Semantic Unification Meta-Model: Volume1 – Semantic unification of static models", ИСО TC184/SC4 WG3 Document N175, October 1992

*) Оригиналы стандартов ИСО (ИСО/МЭК) – во ВНИИКИ Госстандарта России.

Предметный указатель

abstract (зарезервированное слово) 12.10
aggregate (зарезервированное слово) 9.2, 12.3
alias (зарезервированное слово) 11.3
and (зарезервированное слово) 12.10
andor (зарезервированное слово) 12.10
array (зарезервированное слово) 12, 12.3
bag (зарезервированное слово) 12, 12.3
binary (зарезервированное слово) 12.3
boolean (зарезервированное слово) 12.3
call (зарезервированное слово) 10.3
const-e (константа) 8.1.6
constant (зарезервированное слово) 8.8, 12.1, 12.7
context (зарезервированное слово) 9.1, 9.3, 10.3, 11.3.5, 12.9.3–12.9.5, F.2
criteria (зарезервированное слово) 9.3.4
end-context (зарезервированное слово) 11.3
end-criteria (зарезервированное слово) 9.4.3
end-model (зарезервированное слово) 11.3.11
end-notes (зарезервированное слово) 9.4.4
end-purpose (зарезервированное слово) 9.4.1
end-realization (зарезервированное слово) 9.5
end-references (зарезервированное слово) 9.4.2
end-schema-data (зарезервированное слово) 11.3.17
end-test-case (зарезервированное слово) 11.3.20
entity (зарезервированное слово) 7.1, 8.7, 12.1, 12.8, 12.10
enumeration (зарезервированное слово) 7.2, 8.6, 12.1, 12.5, 12.6, 12.9
false (константа) 8.1
function (зарезервированное слово) 12.1
generic (зарезервированное слово) 9.2
import (зарезервированное слово) 10.3
integer (зарезервированное слово) 12.2, 12.10
list (зарезервированное слово) 12.1, 12.2
logical (зарезервированное слово) 12.2
model (зарезервированное слово) 8.10, 11.3.11, 12.9.3–12.9.5
notes (зарезервированное слово) 9.4.4
number (зарезервированное слово) 12.2, 12.10
objective (зарезервированное слово) 9.4
oneof (зарезервированное слово) 12.10
optional (зарезервированное слово) 12.3, 12.9
parameter (зарезервированное слово) 9.2
pi (константа) 8.1.6
procedure (зарезервированное слово) 12, 12.1
purpose (зарезервированное слово) 9.4.1
query (зарезервированное слово) 11.3.14
real (зарезервированное слово) 12.2, 12.10
realization (зарезервированное слово) 9.5
reference (зарезервированное слово) 12.1.1
references (зарезервированное слово) 9.4.2
repeat (зарезервированное слово) 11.3.15
rule (зарезервированное слово) 11.3, 12, 12.1
schema (зарезервированное слово) ..... 9.1, 9.3, 10.2, 11.2, 12.1, приложение F
schema-data (зарезервированное слово) 8.9, 11.3.17
select (зарезервированное слово) 7.3, 8.5, 12.1, 12.6, 12.9
set (зарезервированное слово) 12, 12.3
string (зарезервированное слово) 12.2
subof (зарезервированное слово) 8.7.2, G.11
subtype (зарезервированное слово) 8.7.2, 12.8, 12.10
supertype (зарезервированное слово) 8.7.2, 12.8, 12.10
supof (зарезервированное слово) 8.7.2, G.11

test-case (зарезервированное слово).....9.3, 10.3, 11.3.20
true (константа).....8.1
type (зарезервированное слово) 7.4, 8.4, 12.1
unique (зарезервированное слово) 12.8
unknown (константа) 8.1.5
use (зарезервированное слово) 10.2, 12.1.1
using (зарезервированное слово).....10.2
where (зарезервированное слово)..... 12.8
with (зарезервированное слово) 10.2, 10.3
видимость.....11
нотация.....D.2
область действия.....11

УДК 656.072:681.3:006.354 ОКС 25.040.40 П87 ОКСТУ 4002

Ключевые слова: автоматизация, средства автоматизации, прикладные автоматизированные системы, промышленные изделия, данные, обмен данными, представление данных, языки, EXPRESS-I, руководство

Редактор *В.П. Огурцов*
Технический редактор *Н.С. Гришанова*
Корректор *В.С. Черная*
Компьютерная верстка *А.С. Юфина*

Изд. лиц. №02354 от 14.07.2000. Сдано в набор 23.11.2000. Подписано в печать 13.03.2001. Усл.печ.л. 9,77. Уч.-изд.л. 9,40.
Тираж 327 экз. С 514. Зак. 260.

ИПК Издательство стандартов, 107076, Москва, Колодезный пер., 14.
Набрано в Издательстве на ПЭВМ
Калужская типография стандартов, 248021, Калуга, ул. Московская, 256.
ПЛР № 040138