ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная технология

АБСТРАКТНАЯ СИНТАКСИЧЕСКАЯ НОТАЦИЯ ВЕРСИИ ОДИН (АСН.1)

Часть 1

Спецификация основной нотации

Издание официальное

БЗ 5—2001/106

ГОССТАНДАРТ РОССИИ
Москва

Предисловие

1 РАЗРАБОТАН Государственным научно-исследовательским и конструкторско-технологическим институтом «ТЕСТ» Министерства Российской Федерации по связи и информатизации

ВНЕСЕН Министерством Российской Федерации по связи и информатизации

2 ПРИНЯТ И ВВЕДЕН В ДЕЙСТВИЕ Постановлением Госстандарта России от 6 сентября 2001 г. № 375-ст

3 Настоящий стандарт содержит полный аутентичный текст международного стандарта ИСО/МЭК 8824-1:1998 «Информационная технология. Абстрактная синтаксическая нотация версии один (АСН.1). Часть 1. Спецификация основной нотации» с Изменением № 1 (1999 г.) и Дополнением № 2 (2000 г.)

4 ВВЕДЕН ВПЕРВЫЕ

Содержание

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Информационная технология

АБСТРАКТНАЯ СИНТАКСИЧЕСКАЯ НОТАЦИЯ ВЕРСИИ ОДИН (АСН.1)

Часть 1

Спецификация основной нотации

Information technology. Abstract Syntax Notation One (ASN.1). Specification of basic notation

Дата введения 2002—01—01

1 Область применения

Настоящий стандарт устанавливает нотацию, называемую абстрактной синтаксической нотацией версии один (АСН.1), которая используется для определения типов данных, значений и ограничений типов данных.

Стандарт определяет:

- ряд простых типов вместе с присвоенными им тегами и обозначения для ссылок на эти типы и задания их значений;
- методы построения новых типов из нескольких базовых и обозначения для определения этих типов, присвоения им тегов и задания их значений;
- наборы символов (через указания других стандартов) для использования в АСН.1,
- ряд полезных типов (используя АСН.1), на которые могут ссылаться пользователи АСН.1.

Нотация АСН.1 может применяться во всех случаях, когда требуется определять абстрактный синтаксис информации. Она, в частности (но не единственно), применяется для протоколов прикладного уровня. Ссылки на нотацию АСН.1 содержатся в других стандартах, определяющих правила кодирования для типов АСН.1.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие стандарты:

ГОСТ 34.971—91 (ИСО 8822—88) Системы обработки информации. Взаимосвязь открытых систем. Спецификация услуг уровня представления для режима с установлением соединения (см. также Рекомендацию МСЭ-Т X.216)

ГОСТ 34.972—91 (ИСО 8823—88) Системы обработки информации. Взаимосвязь открытых систем. Спецификация протокола уровня представления для режима с установлением соединения (см. также Рекомендацию МСЭ-Т X.226)

ГОСТ Р ИСО/МЭК 7498-1—99 Информационная технология. Взаимосвязь открытых систем. Базовая эталонная модель. Часть 1. Базовая модель (см. также Рекомендацию МСЭ-Т X.200)

ГОСТ Р ИСО/МЭК 8824—93 Информационная технология. Взаимосвязь открытых систем. Абстрактно-синтаксическая нотация версии 1 (АСН.1)

ГОСТ Р ИСО/МЭК 8824-2—2001. Информационная технология. Абстрактная синтаксическая нотация версии 1 (АСН.1). Часть 2. Спецификация информационного объекта (см. также Рекомендацию МСЭ-Т X.681)

Издание официальное

1-2—2112

ИСО/МЭК 646—91* Информационная технология. 7-битный кодовый набор символов ИСО для информационного обмена

ИСО/МЭК 2022—94* Информационная технология. Структура кода символов и методы расширения

ИСО/МЭК 8824-3—98* Информационная технология. Абстрактная синтаксическая нотация версии 1 (АСН.1). Часть 3. Спецификация ограничения (см. также Рекомендацию МСЭ-Т X.682)

ИСО/МЭК 8824-4—98* Информационная технология. Абстрактная синтаксическая нотация версии 1 (АСН.1). Часть 4. Параметризация спецификаций АСН.1 (см. также Рекомендацию МСЭ-Т X.683)

ИСО/МЭК 8825-1—98* Информационная технология. Правила кодирования АСН.1. Спецификация базовых (BER), канонических (CER) и отличительных (DER) правил кодирования (см. также Рекомендацию МСЭ-Т X.690)

ИСО/МЭК 8825-2—98* Информационная технология. Правила кодирования АСН.1. Спецификация упаковывающих правил кодирования (PER) (см. также Рекомендацию МСЭ-Т X.691)

ИСО/МЭК 9834-1—93* Информационная технология. Взаимосвязь открытых систем. Процедуры работы полномочных органов регистрации ВОС. Часть 1. Общие процедуры (см. также Рекомендацию МСЭ-Т X.660)

ИСО 6523—84* Обмен данными. Структура идентификаторов организаций

ИСО 8601—88* Элементы данных и форматы обмена. Информационный обмен. Представление дат и времени

ИСО/МЭК 10646-1—93* Информационная технология. Универсальный, многооктетный кодовый набор символов (UCS). Часть 1. Архитектура и основная многоязычная плоскость

Рекомендация МСЭ-Т Т. 61 (1988) Репертуар символов и кодовые наборы символов для международных услуг телетекса

Рекомендация МСЭ-Т Т. 100 (1988) Международный информационный обмен для интерактивного видеотекса

Рекомендация МСЭ-Т Т. 101 (1994) Международное межсетевое взаимодействие и для услуг видеотекса

3 Определения

3.1 Спецификация информационного объекта

В настоящем стандарте используют следующие термины, определенные в ГОСТ Р ИСО/МЭК 8824-2:

а) **информационный объект;**

б) **класс информационных объектов;**

в) **набор информационных объектов;**

г) **экземпляр типа;**

д) **тип поля класса объектов.**

3.2 Спецификация ограничения

В настоящем стандарте используют следующие термины, определенные в ИСО/МЭК 8824-3:

а) **ограничение связи компонентов;**

б) **табличное ограничение.**

3.3. Спецификация параметризации АСН.1

В настоящем стандарте используют следующие термины, определенные в ИСО/МЭК 8824-4:

а) **параметризованный тип;**

б) **параметризованное значение.**

3.4 Определение услуг уровня представления

В настоящем стандарте используют следующие термины, определенные в ГОСТ 34.971:

а) **абстрактный синтаксис;**

б) **имя абстрактного синтаксиса;**

в) **множество определенных контекстов;**

г) **значение данных уровня представления;**

---

* Оригиналы и проекты международных стандартов — во ВНИИКИ Госстандарта России.

д) **синтаксис передачи;**
е) **имя синтаксиса передачи.**

3.5 Спецификация протокола уровня представления

В настоящем стандарте используют следующий термин, определенный в ГОСТ 34.972:

- **идентификатор контекста представления.**

3.6 Структура для идентификации организаций

В настоящем стандарте используют следующие термины, определенные в ИСО 6523:

а) **выпускающая организация;**
б) **код организации;**
в) **Международный кодовый определитель** (International Code Designator – ICD).

3.7 Универсальный многооктетный кодовый набор символов (UCS)

В настоящем стандарте используют следующие термины, определенные в ИСО/МЭК 10646-1:

а) **основная многоязычная плоскость** (Basic Multilingual Plane — BMP);
б) **ячейка;**
в) **комбинированный символ;**
г) **графический символ;**
д) **группа;**
е) **ограниченное подмножество;**
ж) **плоскость;**
и) **строка;**
к) **выбранное подмножество.**

3.8 Дополнительные определения

3.8.1 **абстрактный символ:** Множество информации, связанное с ячейкой в таблице, определяющей репертуар символов.

Примечание — Информация обычно включает в себя:

а) графический символ,
б) имя символа или
в) определение функций, связанных с символом при использовании в конкретном окружении.

3.8.2 **абстрактное значение:** Значение, определение которого основывается только на типе, независимо от его представления в любых правилах кодирования.

Примечание — Термин «абстрактное значение» часто используется в утверждениях, которые, вероятно, могут изменяться для различных используемых правил кодирования.

3.8.3 **набор символов АСН.1:** Набор символов, определенный в разделе 10, который используется в нотации АСН.1.

3.8.4 **спецификация АСН.1:** Совокупность одного или нескольких модулей АСН.1.

3.8.5 **ассоциированный тип:** Тип, который используется только для определения значения и нотации подтипа для типа.

Примечание — Ассоциированные типы определены в настоящем стандарте, когда необходимо сделать очевидным, что может быть существенное различие между тем, как тип определен в АСН.1 и как он кодируется. Ассоциированные типы не появляются в спецификациях пользователей.

3.8.6 **тип «битовая строка»:** Простой тип, различными значениями которого являются упорядоченные последовательности из нуля, одного или нескольких бит.

Примечание — Когда необходимо передать встроенное кодирование абстрактного значения, использование типа «встроенное-здп» в общем случае предоставляет более гибкий метод для объявления или согласования характера кодирования, чем битовая строка.

3.8.7 **булевский тип:** Простой тип с двумя различными значениями.

3.8.8 **символ:** Член набора элементов, используемых для организации, управления или представления данных.

Примечание — Например, это подразумевает, что знак ударения и строчная 'e' — два разных символа во французской версии ИСО 646, а не единственный символ е.

3.8.9 **символьный абстрактный синтаксис:** Любой абстрактный синтаксис, значения которого специфицированы как набор символьных строк из нуля, одного или нескольких символов из некоторой заданной совокупности символов.

3.8.10 **репертуар символов:** Символы в наборе символов без какой-либо связи с их кодированием.

3.8.11 **типы символьных строк:** Простые типы, значениями которых являются строки символов из некоторого определенного набора символов.

3.8.12 **символьный синтаксис передачи:** Любой синтаксис передачи для символьного абстрактного синтаксиса.

Примечание — АСН.1 не поддерживает символьные синтаксисы передачи, которые не кодируют все символьные строки как кратные 8 бит.

3.8.13 **выборочные типы:** Типы, определяемые указанием списка различных типов; каждое значение выборочного типа происходит из значения одного из типов-компонентов.

3.8.14 **тип компонента:** Один из типов, указанных при определении CHOICE, SET, SEQUENCE, SET OF или SEQUENCE OF.

3.8.15 **ограничение:** Нотация, которая может быть использована вместе с типом для определения подтипа этого типа.

3.8.16 **управляющие символы:** Символы, появляющиеся в некоторых репертуарах символов, которым должно быть дано имя (и, возможно, определена функция относительно определенных окружений), но которым не присвоен графический символ и которые не являются символом пробела.

Примечание — NEWLINE и TAB являются примерами управляющих символов, которым назначены функции форматирования в среде печати. DLE является примером управляющего символа, которому назначена функция в окружении коммуникации.

3.8.17 **Всемирное согласованное время** (UTC): Шкала времени, поддерживаемая Международным бюро времени и служащая основой для согласованного распространения стандартных частот и сигналов времени.

Примечания

1 Источником этого определения является Рекомендация 460—2 Международной консультативной комиссии по радио (CCIR). Акроним UTC для всемирного согласованного времени также был введен CCIR.

2 UTC и среднее гринвичское время являются двумя альтернативными стандартами времени, которые для большинства практических задач определяют одно и то же время.

3.8.18 **элемент:** Член класса элементов, отличный от всех других элементов этого класса.

3.8.19 **класс элементов:** Тип (элементами которого являются его значения) или информационный объект (элементами которого являются все возможные объекты этого класса).

3.8.20 **набор элементов:** Один или несколько элементов одного и того же класса элементов.

3.8.21 **тип «встроенное-здп»:** Тип, множество значений которого является объединением множеств значений во всех возможных абстрактных синтаксисах. Этот тип является частью спецификации АСН.1 и представляет значение, тип которого может быть определен внешне для данной спецификации АСН.1. Он также представляет идентификацию типа передаваемого значения и идентификацию правил кодирования, использованных для кодирования значения.

3.8.22 **закодированный:** Битовый результат применения правил кодирования к значению данного абстрактного синтаксиса.

3.8.23 **правила кодирования** (АСН.1): Правила, определяющие представление при передаче значений типов АСН.1. Правила кодирования позволяют получателю возможность распознать переданную информацию, предоставляя знания о типе.

Примечание — Для целей спецификации правил кодирования нотации различных указываемых типов (и значений), которые могут предусматривать альтернативные нотации для встроенных типов (и значений), не существенны.

3.8.24 **перечислимые типы:** Простые типы, значения которых задаются различными идентификаторами как часть нотации типа.

3.8.25 **расширяющее дополнение:** Одна из дополнительных нотаций в серии расширений. Для типов "множество", "последовательность" и для выборочного типа каждое расширяющее дополнение является добавлением либо одной расширяющей дополнительной группы, либо одного типа компонента. Для перечислимых типов оно является добавлением одного перечисления. Для ограничения оно является добавлением подтипа элемента.

Примечание — Расширяющие дополнения упорядочены как текстуально (следом за маркером расширения), так и логически (имеют возрастающие перечислимые значения и, в случае альтернатив CHOICE, возрастающие теги).

3.8.26 **расширяющая дополнительная группа:** Один или несколько компонентов типов "множество", "последовательность" или выборочного типа, объединенные скобками версии. Расширяющая дополнительная группа используется для ясной идентификации компонентов множества, последовательности или выбора, которые были добавлены в конкретной версии модуля АСН.1.

3.8.27 **расширяющий дополнительный тип:** Тип, содержащийся в расширяющей дополнительной группе, или единственный тип компонента, который сам является расширяющим дополнением (в этом случае он не содержится в расширяющей дополнительной группе).

3.8.28 **расширяемое ограничение:** Ограничение подтипа с маркером расширения.

3.8.29 **точка вставки расширения:** Место в определении типа, куда вставляются расширяющие дополнения. Это место является концом нотации типа, непосредственно предшествующего типу в серии расширений, если в определении типа имеется единственное многоточие, или находится непосредственно перед вторым многоточием, если в определении типа имеется пара маркеров расширения.

3.8.30 **маркер расширения:** Синтаксический признак (многоточие), включаемый во все типы, образующие серию расширений.

3.8.31 **пара маркеров расширения:** Два маркера расширения, между которыми вставлены расширяющие дополнения.

3.8.32 **связанные расширением:** Два типа, имеющие один и тот же корень расширения, один из которых создан путем добавления к другому нуля или нескольких расширяющих дополнений.

3.8.33 **корень расширения:** Расширяемый тип, который является первым типом в серии расширений. Он имеет либо маркер расширения без дополнительной нотации, отличной от комментариев и пропусков между маркером расширения и завершающей скобкой "}" или ")", либо пару маркеров расширения без дополнительной нотации, отличной от запятой, комментариев и пропусков между ними.

Примечание — Голько корень расширения может быть первым типом в серии расширений.

3.8.34 **серия расширений:** Последовательность типов АСН.1, которые могут быть упорядочены таким образом, что каждый последующий тип образован добавлением текста в точке вставки расширения.

Примечание — Могут быть расширены как вложенные, так и невложенные типы.

3.8.35 **расширяемый тип:** Тип с маркером расширения.

3.8.36 **внешняя ссылка:** Ссылка на тип, значение, информационный объект и пр., определенные в некотором другом модуле, имя которого указывается в виде префикса к указываемому элементу.

Пример — ModuleName. TypeReference

3.8.37 **внешний тип:** Тип, который является частью спецификации АСН.1, представляющий значение, тип которого может быть определен внешне для данной спецификации АСН.1. Он также предоставляет идентификацию типа, значение которого представляется.

3.8.38 **ложно:** Одно из двух различающихся значений булевского типа (см. "истинно").

3.8.39 **управляющий (тип):** Определение типа или ссылка на тип, который влияет на интерпретацию части синтаксиса АСН.1, устанавливая, что часть синтаксиса АСН.1 относится к значениям в управляющем типе.

3.8.39 bis **определения идентичных типов:** Два экземпляра продукции АСН.1 "Type" (см. раздел 16), являющиеся определениями идентичных типов, если, после осуществления преобразований, указанных в приложении F, они являются идентично упорядоченными списками элементов АСН.1 (см. раздел 11).

3.8.40 **целочисленный тип:** Простой тип, различные значения которого являются всеми положительными и отрицательными целыми числами, включая нуль (как одно значение).

Примечание — Конкретные правила кодирования ограничивают диапазон целых чисел, но эти ограничения выбраны так, чтобы не сказываться на пользователях АСН.1.

3.8.41 **элементы:** Поименованные последовательности символов из набора АСН.1, определенного в разделе 9, которые используются для образования нотации АСН.1.

3.8.42 **модуль:** Один или несколько экземпляров использования нотации АСН.1 для типа, значения и т. п., объединенных с использованием нотации модуля АСН.1 (см. раздел 12).

3.8.43 **вырожденный тип:** Простой тип, состоящий из единственного значения, также называемого null.

3.8.44 **объект:** Строго определенная порция информации, определения или спецификации, которой требуется имя для идентификации ее использования в конкретном соединении.

3.8.45 **тип "описатель объекта":** Тип, различными значениями которого является человекочитаемый текст, предоставляющий краткое описание объекта.

Примечание — Значение описателя объекта обычно связано с единственным объектом. Недвусмысленно идентифицирует объект только значение идентификатора объекта.

3.8.46 **идентификатор объекта:** Значение (отличное от всех других таких значений), которое связано с объектом.

3.8.47 **тип "идентификатор объекта":** Простой тип, различные значения которого образуют множество всех идентификаторов объектов, выделенных в соответствии с правилами настоящего стандарта.

Примечание — Правилами ГОСТ Р ИСО/МЭК 9834-1 допускается широкий диапазон уполномоченных для независимого связывания идентификаторов объектов с объектами.

3.8.48 **тип "строка октетов":** Простой тип, различные значения которого являются упорядоченными последовательностями из нуля, одного или нескольких октетов, каждый из которых, в свою очередь, является упорядоченной последовательностью из восьми бит.

3.8.49 **нотация открытого типа:** Нотация АСН.1, используемая для обозначения множества значений из более чем одного типа АСН.1.

Примечания

1 В настоящем стандарте термин «открытый тип» используется как синоним термина «нотация открытого типа».

2 Все правила кодирования АСН.1 обеспечивают недвусмысленное кодирование значений одного типа АСН.1. Они не обязательно обеспечивают недвусмысленное кодирование "нотации открытого типа", представляющей значения из типов АСН.1, которые обычно не являются определенными на момент спецификации. До того, как абстрактное значение для этого поля может быть недвусмысленно определено, необходимо знание типа кодируемого значения.

3 Единственной нотацией открытого типа в настоящем стандарте является "ObjectClassFieldType", определенная в ГОСТ Р ИСО/МЭК 8824-2, где "FieldName" обозначает либо поле типа, либо поле значения переменной-типа. Нотация "ANY", которая определена в ГОСТ Р ИСО/МЭК 8824, была нотацией открытого типа.

3.8.50 **порождающий тип (подтипа):** Тип, который был ограничен при определении подтипа и который управляет нотацией подтипа.

Примечание — Порождающий тип сам может быть подтипом другого типа.

3.8.51 **продукция:** Часть формальной нотации, используемая для спецификации АСН.1.

3.8.52 **действительный тип:** Простой тип, различные значения которого (заданные в разделе 20) являются членами множества действительных чисел.

3.8.53 **рекурсивное определение (типа):** Множество определений АСН.1, которое не может быть упорядочено таким образом, чтобы все типы, используемые в конструкции, были определены до определения самой конструкции.

Примечание — В АСН.1 допускаются рекурсивные определения: пользователь нотации должен гарантировать, чтобы используемые значения (получающихся типов) имели конечные представления.

3.8.54 **ограниченный тип символьных строк:** Тип строки символов, которые берутся из фиксированного репертуара символов, идентифицированного в спецификации типа.

3.8.55 **селективные типы:** Типы, определенные указанием типа компонента выборочного типа, значениями которых являются именно значения этого типа компонента.

3.8.56 **типы "последовательность":** Типы, определенные указанием упорядоченного списка типов (некоторые из них могут быть объявлены как факультативные); каждое значение типа <последовательность> является упорядоченным списком значений, по одному из каждого типа компонентов.

Примечание — Когда тип компонента объявлен факультативным, значение типа «последовательность» не обязательно содержит значение этого типа компонента.

3.8.57 **типы "последовательность-из":** Типы, определенные указанием единственного типа компонента, каждое значение типа "последовательность-из" является упорядоченным списком нуля, одного или нескольких значений этого типа компонента.

3.8.58 **типы «множество»:** Типы, определенные указанием фиксированного, неупорядоченного списка различных типов (некоторые из них могут быть объявлены как факультативные); каждое значение типа "множество" является неупорядоченным списком значений, по одному из каждого типа компонентов.

Примечание — Когда тип компонента объявлен факультативным, значение типа "множество" не обязательно содержит значение этого типа компонента.

3.8.59 **типы "множество-из":** Типы, определенные указанием единственного типа компонента; каждое значение типа «множество-из» является неупорядоченным списком нуля, одного или нескольких значений этого типа компонента.

3.8.60 **простые типы:** Типы, определенные непосредственно спецификацией множеств их значений.

3.8.61 **символ интервала:** Символ в репертуаре символов, который предназначен для включения вместе с графическими символами в печатное представление строки символов, но который в физическом исполнении представляется в виде пустого места; обычно он не рассматривается как управляющий символ (см. 3.8.16).

Примечание — В репертуаре символов может быть единственный символ интервала, а может быть и несколько разной ширины.

3.8.62 **подтип (порождающего типа):** Тип, значения которого являются подмножеством (или полным подмножеством) значений некоторого другого (порождающего) типа.

3.8.63 **тег:** Обозначение типа, которое связывается с каждым типом АСН.1.

3.8.64 **тегированные типы:** Типы, определенные указанием одного существующего типа и тега; новый тип изоморфен существующему типу, но отличается от него.

3.8.65 **тегирование:** Замена существующего (возможно, по умолчанию) тега типа заданным тегом.

3.8.66 **истинно:** Одно из двух различающихся значений булевского типа (см. "ложно").

3.8.67 **тип:** Поименованное множество значений.

3.8.68 **ссылочное имя типа:** Имя, однозначно связанное с типом в некотором контексте.

Примечание — Ссылочные имена присвоены типам, определенным в настоящем стандарте; они всеобще доступны в пределах АСН.1. Другие ссылочные имена определяются в других стандартах; они доступны только в контекстах этих стандартов.

3.8.69 **неограниченный тип символьных строк:** Тип, значениями которого являются значения из символьного абстрактного синтаксиса, идентифицируемого отдельно для каждого случая использования этого типа.

3.8.70 **пользователь (АСН.1):** Лицо или организация, которое определяет абстрактный синтаксис конкретного вида информации, используя АСН.1.

3.8.71 **значение:** Отдельный член множества значений.

3.8.71 bis **отображение значения:** Взаимоотношение 1—1 между двумя типами, которое позволяет ссылку на значения одного из них использовать как ссылку на значения другого. Это может быть использовано, например, при спецификации подтипов и значений по умолчанию (см. приложение F).

3.8.72 **ссылочное имя значения:** Имя, однозначно связанное со значением в некотором контексте.

3.8.73 **множество значений:** Совокупность значений типа. Семантически эквивалентно подтипу.

3.8.74 **скобки версии:** Две пары смежных левых и правых квадратных скобок ([[и]]), используемые в начале и конце расширяющей дополнительной группы.

3.8.75 **пропуск:** Любое действие форматирования, оставляющее пустое место на печатной странице, как символы SPACE или TAB, или несколько подряд таких символов.

4 Сокращения

АСН.1 — абстрактно-синтаксическая нотация версии 1
ЗДП — значение данных представления

BMP – основная многоязычная плоскость (Basic Multilingual Plane)
BER – базовые правила кодирования (Basic Encoding Rules)
CER – канонические правила кодирования (Canonical Encoding Rules)
DER – отличительные правила кодирования (Distinguished Encoding Rules)
PER – упаковывающие правила кодирования (Packed Encoding Rules)
UCS – универсальный, многооктетный кодовый набор символов (Universal Multiple-Octet Coded Character Set)
UTC – всемирное согласованное время (Coordinated Universal Time)
ICD – Международный кодовый определитель (International Code Designator)
DCC – цифровой код страны (Data Country Code)
DNIC – идентификационный код сети передачи данных (Data Network Identification Code)

5 Обозначения, используемые в настоящем стандарте

Нотация АСН.1 состоит из последовательностей символов набора АСН.1, определенного в разделе 10.

Каждый конкретный случай использования нотации АСН.1 содержит символы из символьного набора АСН.1, сгруппированные в элементы. В разделе 11 определяются все последовательности символов, образующие элементы АСН.1, и каждому элементу присваивается имя.

Нотация АСН.1 определяется в разделе 12 (и последующих разделах) путем спецификации совокупности последовательностей элементов, которые образуют допустимые экземпляры АСН.1, и спецификации семантики каждой такой последовательности.

Для того чтобы специфицировать эту совокупность, в настоящем стандарте используют формальную нотацию, определяемую в последующих подразделениях.

5.1 Продукции

Новая (более сложная) совокупность последовательностей определяется с помощью продукции. Она использует имена совокупностей последовательностей продукций, определенных в настоящем стандарте, и строит новую совокупность последовательностей продукций, специфицируя либо:

а) что новая совокупность из последовательностей продукций должна состоять из любой последовательности, содержащейся в любой из исходных совокупностей, либо

б) что новая совокупность должна состоять из любой последовательности продукций, которая может быть образована, если взять ровно по одной последовательности продукций из каждой исходной совокупности и соединить их в заданном порядке.

Каждая продукция состоит из следующих частей, в одну или несколько строк, в следующем порядке:

1) имя новой совокупности последовательностей продукций;

2) символ : : =

3) одна или несколько альтернативных совокупностей последовательностей продукций, определенных в 5.2 и разделенных знаком |

Последовательность продукций присутствует в новой совокупности, если она присутствует в одной или нескольких альтернативных совокупностях. Ссылки на новую совокупность в настоящем стандарте осуществляются по имени из перечисления 1).

Примечание — Если одна и та же последовательность продукций появляется более чем в одной альтернативе, то любая появляющаяся семантическая двусмысленность снимается другими частями полной последовательности продукций АСН.1.

5.2 Альтернативные совокупности

Каждая альтернативная совокупность последовательностей продукций в одной или нескольких альтернативных совокупностях (см. 5.1в) задается списком имен. Каждое имя является либо именем элемента, либо именем совокупности последовательностей продукций, определенной с помощью продукции в настоящем стандарте.

Совокупность последовательностей продукций, определяемая альтернативой, состоит из всех последовательностей продукций, которые получаются, если взять любую из последовательностей продукций (или элемент), связанных с первым именем, за ней — любую из последовательностей продукций (или элемент), связанных со вторым именем, за ней — любую из последовательностей

продукций (или элемент), связанных со третьим именем, и так далее до последнего имени (или элемента) в альтернативе включительно.

5.3 Пример продукции

BitStringValue : : = bstring |
hstring | "{" IdentifierList "}"

Эта запись является продукцией, которая связывает с именем <BitStringValue> следующие последовательности продукций:

а) любая "bstring" (элемент), или

б) любая "hstring" (элемент), или

в) любая последовательность продукций, связанная с "IdentifierList", перед которой стоит знак {, а за которой — знак }.

Примечание — "{" и "}" являются именами элементов, состоящими из одного знака, соответственно { и } (см. 11.17).

В данном примере "IdentifierList" должно быть определено другой продукцией, либо до, либо после продукции "BitStringValue".

5.4 Размещение текста

Перед и после каждой продукции, используемой в настоящем стандарте, стоит пустая строка. Внутри продукции пустые строки отсутствуют. Продукция может либо располагаться в одной строке, либо разбиваться на несколько строк. Размещение текста продукции не имеет значения.

5.5 Рекурсия

Продукции, используемые в настоящем стандарте, часто являются рекурсивными. В таком случае продукции применяются до тех пор, пока не будут сгенерированы новые последовательности.

Примечание — Во многих случаях такое повторение приводит к неограниченной совокупности допустимых последовательностей, некоторые или все из которых сами могут быть неограниченными. Это не является ошибкой.

5.6 Ссылки на совокупность последовательностей

В настоящем стандарте ссылка на совокупность последовательностей (часть нотации АСН.1) осуществляются указанием первого имени в продукции (перед : : =); имя окружается двойными кавычками ("), чтобы отличать от текста на русском языке; в продукции такое выделение не делается.

5.7 Ссылки на элемент

В настоящем стандарте ссылки на элемент осуществляются указанием имени элемента; для того чтобы отличать имя от текста на естественном языке, оно заключается в двойные кавычки ("), если только оно не является частью продукции и не является односимвольным элементом, " : : = ", " . . " или " . . . ".

5.8 Краткие формы нотации

Для того чтобы сделать продукции более сжатыми и более удобочитаемыми, следующие краткие формы нотации используются в определении совокупностей последовательностей продукций АСН.1 в ГОСТ Р ИСО/МЭК 8824-2, ИСО/МЭК 8824-3 и ИСО/МЭК 8824-4 (в настоящем стандарте они не используются):

а) звездочка (*) следом за двумя именами ("А" и "В") обозначает либо пустой элемент (см. 11.7), либо последовательность продукций, ассоциированную с "А", либо альтернативные серии последовательностей продукций, связанных с "А" и "В", начинающиеся и заканчивающиеся последовательностями, ассоциированными с "А". Так,

C : : = A B *

эквивалентно

C : : = D | empty
D : : = A | A B D

"D" является вспомогательным именем, нигде в продукциях не появляющимся.

Пример — "C : : = A B *" является краткой формой нотации для следующих альтернатив С:

empty
A
A B A
A BA B A
A BA B A B A
...

б) знак плюс (+) аналогичен звездочке из перечисления а), за исключением того, что пустой элемент не допускается. Так,

E : : = A B +

эквивалентно

E : : = A | A B E

Пример – "E : : = A B +" является краткой формой нотации для следующих альтернатив E:

A
A B A
A BA B A
A BA B A B A
...

в) знак вопроса (?) следом за именем обозначает либо пустой элемент (см. 11.7), либо последовательность продукций, ассоциированных с "A". Так,

F : : = A ?

эквивалентно

F : : = empty | A

5.9 Ссылки на значения и типиазация значений

5.9.1 АСН.1 определяет нотацию присваивания значений, которая позволяет дать имя значению специфицированного типа. Это имя может использоваться всякий раз, когда нужна ссылка на значение. В приложении F описан и установлен метод отображения значений, который позволяет ссылочному имени значения одного типа идентифицировать значение другого типа. Таким образом, ссылка на первое значение может использоваться тогда, когда требуется ссылка на значение второго типа.

5.9.2 В стандартах АСН.1 используется описание на обычном языке для спецификации допустимости (или недопустимости) конструкций, в которых участвует несколько типов, но при этом два типа должны быть «совместимы». Например требуется, чтобы тип, используемый при определении ссылки на значение, был «совместим» с управляющим типом при использовании этой ссылки. В приложении F понятие отображения значений используется для точного утверждения о том, допустима данная конструкция АСН.1 или нет.

6 Модель АСН.1 расширения типа

При декодировании расширяемого типа декодер может выявить:

а) отсутствие ожидаемых расширяющих дополнений в типе "последовательность" или "множество", или

б) присутствие либо произвольных неожидаемых расширяющих дополнений, кроме тех, которые определены (если были определены) в типе "последовательность" или "множество", либо неизвестной альтернативы в выборочном типе, либо неизвестного перечисления в перечислимом типе, либо неожиданной длины или значения типа, ограничение которого расширяется.

Формально говоря, абстрактный синтаксис, определяемый расширяемым типом "X", содержит не только значения типа "X", но и значения всех типов, связанных расширением с "X". Таким образом, процесс декодирования никогда не сигнализирует об ошибке при выявлении любой из перечисленных ситуаций а) и б). Действия, которые должны быть предприняты в такой ситуации, должны быть специфицированы проектировщиком прикладного уровня.

Примечание — Часто действием должно быть игнорирование присутствия неожидаемых дополнительных расширений и использование значений по умолчанию или указателей "отсутствует" для отсутствующих, но ожидаемых расширяющих дополнений.

Неожидаемые расширяющие дополнения, выявленные декодером в расширяемом типе, могут быть позже включены в последующие кодирования этого типа (для передачи обратно отправителю или какой-либо третьей стороне), при условии, что для последующей передачи используется тот же самый синтаксис передачи.

7 Требования расширяемости для правил кодирования

7.1 Все правила кодирования АСН.1 должны допускать кодирование значения расширяемого типа "X" таким образом, что оно может быть декодировано с использованием расширяемого типа "Y", который связан расширением с "X". Более того, правила кодирования должны допускать, чтобы значения, декодированные с использованием "Y", были повторно закодированы (используя "Y") и декодированы с использованием третьего расширяемого типа "Z", который связан расширением с "Y" (и, следовательно, с "X").

Примечание — Типы "X", "Y" и "Z" могут появляться в последовательности расширения в любом порядке.

Если значение расширяемого типа "X" закодировано и передано (непосредственно или через ретранслирующее приложение, использующее связанный расширением тип "Z") другому приложению, которое декодирует значение, используя расширяемый тип "Y", связанный расширением с "X", то декодер, использующий тип "Y", получит абстрактное значение, составленное из:

а) абстрактного значения типа корня расширения;

б) абстрактного значения каждого расширяющего дополнения, которое присутствует как в "X", так и в "Y";

в) выделенного кодирования для каждого расширяющего дополнения, которое есть в "X", но отсутствует в "Y" (если такие имеются).

Кодирование в случае в) должно быть пригодно для включения в последующее кодирование значения "Y", если это требуется приложением. Это кодирование должно быть допустимым кодированием значения "X".

Пример — Если система А использует тип расширяемого корня (тип "X"), являющийся типом "последовательность" или "множество" с расширяющим дополнением факультативного целого типа, а система В использует связанный расширением тип (тип "Y"), имеющий два расширяющих дополнения, каждое из которых — факультативный целый тип, то при передаче от В значения "Y" с опущенным целым значением первого расширяющего дополнения и включенным вторым, оно не должно быть перепутано системой А с наличием первого (единственного) расширяющего дополнения "X", о котором ей известно. Более того, А должна быть в состоянии повторно закодировать значение "X" со значением, предоставленным для первого целого типа, с последующим вторым целым значением, полученным от В, если это требуется прикладным протоколом.

7.2 Все правила кодирования АСН.1 должны специфицировать кодирование и декодирование значения перечислимого и выборочного типов таким образом, что если передаваемое значение — из множества расширяющих дополнений, общего для кодера и декодера, то оно будет успешно декодировано, в противном случае у декодера должна быть возможность выделить кодирование этого значения и идентифицировать его как значение (неизвестного) расширяющего дополнения.

7.3 Все правила кодирования АСН.1 должны специфицировать кодирование и декодирование типов с расширяемыми ограничениями таким образом, что если передаваемое значение — из множества расширяющих дополнений, общего для кодера и декодера, то оно будет успешно декодировано, в противном случае у декодера должна быть возможность выделить кодирование этого значения и идентифицировать его как значение (неизвестного) расширяющего дополнения.

В любом случае присутствие расширяющих дополнений не должно влиять на возможность распознавания последующего материала, когда тип с маркером расширения вложен в некоторый другой тип.

Примечание — Все варианты базовых и упаковывающих правил кодирования АСН.1 удовлетворяют всем этим требованиям.

8 Теги

8.1 Тег задается указанием его класса и номера в классе. Определены следующие классы тегов:

- универсальный (universal);
- прикладной (application);
- пользовательский (private);
- контекстно зависимый (context — specific).

8.2 Номер является неотрицательным целым числом, заданным в десятичной нотации.

Ограничения на теги, присваиваемые пользователем АСН.1, определены в разделе 32.

В таблице 1 приведена сводка тегов универсального класса, присвоенных в настоящем стандарте.

8.3 Некоторые правила кодирования требуют канонического порядка тегов. Для обеспечения единообразия канонический порядок тегов определен в 8.4.

8.4 Канонический порядок тегов определяется следующим образом:

а) первыми должны появляться элементы или альтернативы с тегами универсального класса, далее — прикладного, затем — контекстно зависимого и последними — с тегами пользовательского класса;

б) в пределах каждого класса элементы или альтернативы должны появляться в порядке возрастания номеров их тегов.

Таблица 1 — Присвоенные теги универсального класса

UNIVERSAL 0	Зарезервировано для использования правилами кодирования
UNIVERSAL 1	Булевский тип
UNIVERSAL 2	Целочисленный тип
UNIVERSAL 3	Тип "битовая строка"
UNIVERSAL 4	Тип "строка октетов"
UNIVERSAL 5	Вырожденный тип
UNIVERSAL 6	Тип "идентификатор объекта"
UNIVERSAL 7	Тип "описатель объекта"
UNIVERSAL 8	Внешний тип и тип "экземпляр-из"
UNIVERSAL 9	Вещественный тип
UNIVERSAL 10	Перечислимый тип
UNIVERSAL 11	Тип "встроенное-зди"
UNIVERSAL 12	Тип UTF8String
UNIVERSAL 13—15	Зарезервировано для последующих редакций стандарта
UNIVERSAL 16	Типы "последовательность" и "последовательность-из"
UNIVERSAL 17	Типы "множество" и "множество-из"
UNIVERSAL 18—22, 25—30	Типы символьных строк
UNIVERSAL 23—24	Типы времени
UNIVERSAL 31— . . .	Зарезервировано для дополнений к стандарту

9 Использование нотации АСН.1

9.1 Нотацией АСН.1 для определения типа должна быть "Type" (см. 16.1).

9.2 Нотацией АСН.1 для определения значения должна быть "Value" (см. 16.7).

Примечание — В общем случае невозможно интерпретировать нотацию значения без знания типа.

9.3 Нотацией АСН.1 для присвоения типа ссылочному имени типа должна быть либо "TypeAssignment" (см.15.1), либо "ValueSetTypeAssignment" (см.15.4), либо "ParameterizedTypeAssignment" (см. ИСО/МЭК 8824-4, 8.2), либо "ParameterizedValueSetTypeAssignment" (см. ИСО/МЭК 8824-4, 8.2).

9.4 Нотацией АСН.1 для присвоения значения ссылочному имени значения должна быть либо "ValueAssignment" (см. 15.2), либо "ParameterizedValueAssignment" (см. ИСО/МЭК 8824-4, 8.2).

9.5 Альтернативы продукции нотации «Assignment» должны использоваться только в нотации "ModuleDefinition" (за исключением, указанным в примечании 2 к 12.1).

10 Набор символов АСН.1

10.1 Каждый элемент АСН.1 состоит из последовательности символов, приведенных в таблице 2, за исключением случаев, указанных в 10.2 и 10.3. В таблице 2 символы идентифицируются именами, данными в ИСО/МЭК 10646-1.

Таблица 2 — Символы АСН.1

От A до Z	(от LATIN CAPITAL LETTER A до LATIN CAPITAL LETTER Z — прописные латинские буквы от A до Z)
От a до z	(от LATIN SMALL LETTER A до LATIN SMALL LETTER Z — строчные латинские буквы от a до z)
От 0 до 9	(от DIGIT ZERO до DIGIT 9 — цифры от нуля до 9)
:	(COLON — двоеточие)
=	(EQUALS SIGN — знак "равно")
,	(COMMA — запятая)
{	(LEFT CURLY BRACKET — левая фигурная скобка)
}	(RIGHT CURLY BRACKET — правая фигурная скобка)
<	(LESS-THAN SIGN — знак "меньше, чем")
.	(FULL STOP — точка)
@	(COMMERCIAL AT — коммерческое "эт")
(	(LEFT PARENTHESIS — левая скобка)
)	(RIGHT PARENTHESIS — правая скобка)
[	(LEFT SQUARE BRACKET — левая квадратная скобка)
]	(RIGHT SQUARE BRACKET — правая квадратная скобка)
—	(HYPHEN-MINUS — дефис)
'	(APOSTROPHE — апостроф)
"	(QUOTATION MARK — кавычка)
|	(VERTICAL LINE — вертикальная черта)
&	(AMPERSAND — амперсанд)
^	(CIRCUMFLEX ACCENT — знак ударения)
*	(ASTERISK — звездочка)
;	(SEMICOLON — точка с запятой)
!	(EXCLAMATION MARK — восклицательный знак)

Примечание — В стандартах, производных от данного стандарта, которые разрабатываются национальными организациями по стандартизации, дополнительные символы могут присутствовать в следующих элементах:

typereference	(см. 11.2);
identifier	(см. 11.3);
valuereference	(см. 11.4);
modulereference	(см. 11.5).

Когда дополнительные знаки вводятся для соответствия алфавитам, в которых различие между строчными и прописными буквами не имеет значения, синтаксическое различие, достигаемое за счет предписания регистра первого символа некоторых из приведенных выше элементов АСН.1, должно обеспечиваться каким-то другим способом. Тем самым допускается написание допустимых спецификаций АСН.1 на разных языках.

10.2 При использовании нотации для спецификации значения типа символьная строка в нотации АСН.1 могут появляться все графические символы для определенного набора символов, взятые в двойные кавычки (") (см. 11.11).

10.3 Дополнительные (произвольные) графические символы могут появляться в элементе "комментарии" (см. 11.6).

10.4 Не следует придавать какого-либо значения типографскому стилю, размеру, цвету, яркости и другим характеристикам отображения.

10.5 Строчные и прописные буквы следует рассматривать как различные знаки.

11 Элементы АСН.1

11.1 Общие правила

11.1.1 В последующих подразделах определяются символы в элементах АСН.1. В каждом случае даются имя элемента и определение символьных последовательностей, которые образуют элемент.

11.1.2 Каждый элемент, определяемый в последующих подразделах (за исключением "bstring", "hstring" и "cstring"), должен располагаться в пределах одной строки и (за исключением элементов "comment", "bstring", "hstring" и "cstring") не должен содержать пробелов (см. 11.9—11.11).

11.1.3 Длина строки не ограничивается.

11.1.4 Элементы в последовательностях продукций, определенные в настоящем стандарте (в обозначениях нотации АСН.1), могут располагаться на одной или нескольких строках и разделяться одним или несколькими пробелами, пустыми строками или комментариями.

11.1.5 Каждый элемент должен быть отделен от следующего за ним элемента пробелом, новой строкой или комментарием, если начальный символ (или символы) следующего элемента являются допустимыми для включения в конец последовательности символов предыдущего элемента.

11.1.6 В настоящем стандарте использованы термины "новая строка", "конец строки", "пропуск". При представлении пропуска и новой строки (конца строки) в машиночитаемых спецификациях могут использоваться любые из следующих символов в любой комбинации (символы названы и идентифицированы десятичными значениями, которые являются значениями кодов символов по ИСО/МЭК 646):

- для пропуска
  - HORIZONTAL TABULATION (9) — горизонтальная табуляция,
  - SPACE (32) — пробел,
  - LINE FEED (10) — пропуск строки,
  - VERTICAL TAB (11) — вертикальная табуляция,
  - FORM FEED (12) — пропуск страницы,
  - CARRIAGE RETURN (13) — возврат каретки;
- для новой строки:
  - LINE FEED (10) — пропуск строки,
  - VERTICAL TAB (11) — вертикальная табуляция,
  - FORM FEED (12) — пропуск страницы,
  - CARRIAGE RETURN (13) — возврат каретки.

11.2 Ссылки на тип

Имя элемента — typereference.

11.2.1 Элемент "typereference" должен состоять из произвольного количества (большего или равного единице) букв, цифр и дефисов. Начальный символ должен быть прописной буквой. Последний символ не должен быть дефисом. Два дефиса не должны следовать друг за другом.

Примечание — Правила использования дефиса должны исключать двусмысленность с (возможным последующим) комментарием.

11.2.2 Элемент "typereference" не должен совпадать с одной из зарезервированных символьных последовательностей, приведенных в 11.18.

11.3 Идентификаторы

Имя элемента — identifier.

Элемент "identifier" должен состоять из произвольного количества (большего или равного единице) букв, цифр и дефисов. Начальный символ должен быть строчной буквой. Последний символ не должен быть дефисом. Два дефиса не должны следовать друг за другом.

Примечание — Правила, относящиеся к использованию дефиса, направлены на то, чтобы избежать двусмысленности с (возможным последующим) комментарием.

11.4 Ссылки на значение

Имя элемента — valuereference.

Элемент "valuereference" должен состоять из последовательности символов, заданных для "identifier" в 11.3. При анализе конкретного использования данной нотации элемент "valuereference" отличается от элемента "identifier" контекстами, в которых они появляются.

11.5 Ссылка на модуль

Имя элемента — modulereference.

Элемент "modulereference" должен состоять из последовательности символов, заданной для элемента "typereference" в 11.2. При анализе конкретного использования данной нотации элемент "modulereference" отличается от элемента "typereference" контекстами, в которых они появляются.

11.6 Комментарий

Имя элемента — comment.

11.6.1 Комментарий не указывается в определении нотации АСН.1. Однако он может появляться в любом месте между другими элементами нотации АСН.1, но не имеет синтаксического значения.

11.6.2 Элемент "comment" должен начинаться с пары следующих друг за другом дефисов и заканчиваться следующей парой таких же дефисов или текущей строкой, в зависимости от того, что встретиться раньше. Комментарий не должен содержать пар следующих друг за другом дефисов, кроме начальных и завершающих (в случае наличия). Комментарии может содержать графические символы, не входящие в символьный набор, определенный в 10.1 (см. 10.3).

11.7 Пустой элемент

Имя элемента — empty.

Элемент "empty" не содержит никаких символов. Он используется в нотации раздела 5, когда заданы альтернативные множества последовательностей продукций для указания на возможное отсутствие всех альтернатив.

11.8 Элемент "число"

Имя элемента — number.

Элемент "number" состоит из одной или нескольких цифр. Первая цифра не может быть нулем, за исключением случая, когда "number" состоит из одной цифры.

Примечание — Элемент "number" всегда отображается в целочисленное значение путем интерпретации его как десятичного обозначения.

11.9 Элемент "двоичная строка"

Имя элемента — bstring.

Элемент "bstring" состоит из произвольного количества (возможно — нулевого) нулей, единиц, пропусков или новых строк, перед которыми стоит одиночная кавычка (') и за которыми следует пара символов

'B

Пропуски и новые строки, которые появляются в двоичной строке, значения не имеют.

Пример — '01101100'B

11.10 Элемент "шестнадцатеричная строка"

Имя элемента — hstring.

11.10.1 Элемент "hstring" состоит из произвольного количества (возможно — нулевого) символов

A B C D E F 0 1 2 3 4 5 6 7 8 9

или пропусков и новых строк, перед которыми стоит одиночная кавычка (') и за которыми следует пара символов

'H

Пропуски и новые строки, которые появляются в шестнадцатеричной строке, значения не имеют.

Пример — 'AB0196'H

11.10.2 Каждый символ используется для обозначения значения полуоктета в шестнадцатеричном представлении.

11.11 Элемент «символьная строка»

Имя элемента — cstring.

11.11.1 Элемент "cstring" состоит из произвольного количества (возможно — нулевого) графических символов и интервалов из набора символов, определяемого типом рассматриваемой строки, перед которыми и после них следуют двойные кавычки ("). Если набор символов содержит двойные кавычки, то этот символ (если он присутствует в символьной строке, представляемой "cstring") должен быть представлен в "cstring" парой двойных кавычек в одной и той же строке, без интервала между ними. "cstring" может занимать несколько строк текста; в этом случае представляемая символьная строка не должна содержать символов интервала в позиции до или после конца строки в "cstring". Пропуски, которые появляются непосредственно до или после конца строки в "cstring", значения не имеют.

Примечания

1 Элемент "cstring" может использоваться только для представления символьных строк, всем представляемым символам которых либо был присвоен графический знак, либо они являются символом интервала. Когда требуется обозначить символьную строку, содержащую управляющие символы, имеется альтернативный синтаксис ACH.1 (см. раздел 34).

2 Символьная строка, представленная "cstring", состоит из символов, связанных с печатными графическими символами и символами интервала. Символы интервала, непосредственно предшествующие концу строки в "cstring" или следующие непосредственно за ним, не являются частью представляемой символьной строки

(они игнорируются). Когда символы интервала входят в "cstring" или графические символы в символьном репертуаре являются двусмысленными, символьная строка, обозначаемая "cstring", может быть двусмысленной.

Пример 1 – 

Пример 2 — Элемент "cstring"

"ABCDE FGH
IJK""XYZ"

может быть использован для представления значения символьной строки типа IA5String. Представленное значение состоит из символов

ABCDE FGHIJK"XYZ

где точное число пробелов между E и F может быть неоднозначным, если в спецификации используется пропорциональный шрифт.

11.11.2 Когда символ является комбинированным, он должен быть обозначен в "cstring" как один символ. Он не должен являться наложением символов, из которых скомбинирован. (Тем самым подразумевается, что порядок комбинирования символов в значении строки определен однозначно).

Пример — Комбинация ударения и строчного `e` является двумя символами во французской версии ИСО 646 и, таким образом, в соответствующей "cstring" записывается как два символа 'e', а не один.

11.11.3 Элемент "cstring" не должен использоваться для представления значений символьных строк, содержащих управляющие символы. В нем допустимы только графические символы и знаки интервала.

11.12 Элемент «присвоение»

Имя элемента – ": : ="

Этот элемент состоит из последовательности символов

: : =

Примечание — Эта последовательность не содержит пробелов (см. 11.1.2).

11.13 Разделитель диапазона

Имя элемента – ". ."

Этот элемент состоит из последовательности символов

..

Примечание — Эта последовательность не содержит пробелов (см. 11.1.2).

11.14 Многоточие

Имя элемента – ". . ."

Этот элемент состоит из последовательности символов

. . .

Примечание — Эта последовательность не содержит пробелов (см. 11.1.2).

11.15 Левые скобки версии

Имя элемента – "[["

Этот элемент должен состоять из последовательности символов

[[

Примечание — Эта последовательность не содержит символов пропуска (см. 11.1.2).

11.16 Правые скобки версии

Имя элемента – "]]"

Этот элемент должен состоять из последовательности символов

]]

Примечание — Эта последовательность не содержит символов пропуска (см. 11.1.2).

11.17 Элементы, состоящие из одного символа

Имена элементов –

"{"

"}"

"<"

","
"."
"("
")"
"["
"]"
"-" (дефис)
":"
"="
"@"
"|"
"!"
"^"

Элемент с любым из перечисленных выше имен состоит из единственного символа без кавычек.

11.18 Зарезервированные слова

Имена зарезервированных слов следующие:

ABSENT	END	INTEGER	SEQUENCE
ABSTRACT-SYNTAX	ENUMERATED	INTERSECTION	SET
ALL	EXCEPT	ISO646String	SIZE
APPLICATION	EXPLICIT	MAX	STRING
AUTOMATIC	EXPORTS	MIN	SYNTAX
BEGIN	EXTENSIBILITY	MINUS-INFINITY	T61String
BIT	EXTERNAL	NULL	TAGS
BMPString	FALSE	NumericString	TeletexString
BOOLEAN	FROM	OBJECT	TRUE
BY	GeneralizedTime	ObjectDescriptor	TYPE – IDENTIFIER
CHARACTER	GeneralString	OCTET	UNION
CHOICE	GraphicString	OF	UNIQUE
CLASS	IA5String	OPTIONAL	UNIVERSAL
COMPONENT	IDENTIFIER	PDV	UniversalString
COMPONENTS	IMPLICIT	PLUS-INFINITY	UTCTime
CONSTRAINED	IMPLIED	PRESENT	UTF8String
DEFAULT	IMPORTS	PrintableString	VideotexString
DEFINITIONS	INCLUDES	PRIVATE	VisibleString
EMBEDDED	INSTANCE	REAL	WITH

Элементы с перечисленными выше именами состоят из последовательностей символов в имени и являются зарезервированными символьными последовательностями.

Примечания

1 В приведенных выше последовательностях символов отсутствуют пробелы.

2 Ключевые слова CLASS, CONSTRAINED, INSTANCE, SYNTAX и UNIQUE не используются в настоящем стандарте; они используются в ГОСТ Р ИСО/МЭК 8824-2, ИСО/МЭК 8824-3 и ИСО/МЭК 8824-4.

12 Определение модуля

12.1 Определение модуля "ModuleDefinition" задается следующими продукциями:

ModuleDefinition : : =
    ModuleIdentifier
    DEFINITIONS
    TagDefault
    ": : ="
    BEGIN
    ModuleBody
    END

```
ModuleIdentifier : : =
        modulereference
        DefinitiveIdentifier
DefinitiveIdentifier : : =
        "{" DefinitiveObj IdComponentList "}" | empty
DefinitiveObj IdComponentList : : =
        DefinitiveObj IdComponent    |
        DefinitiveObj IdComponent DefinitiveObj IdComponentList
DefinitiveObj IdComponent : : =
        NameForm                |
        DefinitiveNumberForm    |
        DefinitiveNameAndNumberForm
DefinitiveNumberForm : : =  number
DefinitiveNameAndNumberForm : : =  identifier "(" DefinitiveNumberForm ")"
TagDefault : : =
        EXPLICIT TAG                |
        IMPLICIT TAG                |
        AUTOMATIC TAGS              |
        empty
ExtensionDefault : : =
        EXTENSIBILITY IMPLIED       |
        empty
ModuleBody : : =
        Exports Imports AssignmentList      |
        empty
Exports : : =
        EXPORTS SymbolsExported ";"     |
        empty
SymbolsExported : : =
        SymbolList          |
        empty
Imports : : =
        IMPORTS SymbolsImported ";"     |
        empty
SymbolsImported : : =
        SymbolsFromModuleList               |
        empty
SymbolsFromModuleList : : =
        SymbolsFromModule                   |
        SymbolsFromModuleList SymbolsFromModule
SymbolsFromModule : : =
        SymbolList FROM GlobalModuleReference
GlobalModuleReference : : =
        modulereference AssignedIdentifier
AssignedIdentifier : : =
        ObjectIdentifierValue       |
        DefinedValue                |
        empty
SymbolList : : =
        Symbol          |
        SymbolList "," Symbol
Symbol : : =
        Reference       |
        ParameterizedReference
```

```
Reference : : =
        typereference                   |
        valuereference                  |
        objectclassreference            |
        objectreference                 |
        objectsetreference
AssignmentList : : =
        Assignment          |
        AssignmentList Assignment
Assignment : : =
        TypeAssignment                  |
        ValueAssignment                 |
        ValueSetTypeAssignment          |
        ObjectClassAssignment           |
        ObjectAssignment                |
        ObjectSetAssignment             |
        ParameterizedAssignment
```

Примечания

1 Использование ParameterizedReference в списках EXPORTS и IMPORTS специфицировано в ИСО/МЭК 8824-4.

2 В примерах (и определениях в настоящем стандарте типов с тегами универсального класса) "ModuleBody" может использоваться вне "ModuleDefinition".

3 Продукции "TypeAssignment" и "ValueAssignment" специфицированы в разделе 15.

4 Группирование типов данных АСН.1 в модули не определяет обязательного формирования значений данных уровня представления в поименованные абстрактные синтаксисы с целью определения контекста представления.

5 Значение "TagDefault" для определения модуля влияет только на типы, явно определенные в модуле. Оно не влияет на интерпретацию импортированных типов.

6 Точка с запятой не ставиться в спецификации списка присвоений "AssignmentList" или в какой-либо подчиненной продукции, а зарезервирована для использования разработчиками инструментов АСН.1.

12.2 Если тег по умолчанию "TagDefault" равен "empty", то подразумевается "EXPLICIT TAGS".

Примечание — Смысл "EXPLICIT TAGS", "IMPLICIT TAGS" и "AUTOMATIC TAGS" определяется в разделе 32.

12.3 Когда выбрана альтернатива "AUTOMATIC TAGS" для "TagDefault", то говорят, что для модуля выбрано автоматическое тегирование. Автоматическое тегирование — это синтаксическое преобразование, которое применяется (с дополнительными условиями) к продукциям "ComponentTypeList" и "AlternativeTypeList", встретившимся в определении модуля. Это преобразование формально специфицировано в подразделах 24.7—24.9, 26.3 и 28.2 для нотаций типов последовательностей, множеств и выборочных типов соответственно.

12.4 Опция "EXTENSIBILITY IMPLIED" эквивалентна вставке в модуле маркера расширения (. . .) в определении каждого типа, для которого это допустимо. Подразумеваемый маркер расширения располагается в типе в последней позиции, в которой допустима явная спецификация маркера расширения. Отсутствие "EXTENSIBILITY IMPLIED" означает, что расширение обеспечивается только для тех типов в модуле, для которых маркер расширения присутствует явно.

Примечание — "EXTENSIBILITY IMPLIED" влияет только на типы, но не влияет на множества объектов.

12.5 Ссылка на модуль "modulereference", появляющаяся в продукции "ModuleIdentifier", называется именем модуля.

Примечание — Возможность определения одного модуля АСН.1 с помощью нескольких элементов "ModuleBody" с присвоением одной и той же "modulereference" допускалась предшествующими спецификациями. Настоящей спецификацией это не допускается.

12.6 Имена модулей должны использоваться только один раз (за исключением, определенным в 12.9) в сфере применения определения модуля.

12.7 Если определяющий идентификатор "DefinitiveIdentifier" не пуст, то значение идентификатора объекта недвусмысленно и однозначно идентифицирует определяемый модуль. Не определенное значение может быть использовано в определяющем значении идентификатора объекта.

Примечание — Вопрос, какие изменения потребуются в модуле для нового "DefinitiveIdentifier", не рассматривается в настоящем стандарте.

12.8 Если присвоенный идентификатор "AssignedIdentifier" не пуст, то альтернативы "ObjectIdentifierValue" и "DefinedValue" недвусмысленно и однозначно идентифицируют модуль, из которого должны быть импортированы элементы. Когда в "AssignedIdentifier" используется альтернатива "DefinedValue", она должна быть значением идентификатора типа объекта. Каждая ссылка на значение "valuereference", текстуально появляющаяся в "AssignedIdentifier", должна удовлетворять одному из следующих правил.

а) ссылка определена в "AssignmentList" определяемого модуля и все "valuereference", текстуально появляющиеся в правой части предложения присваивания, также удовлетворяют этому (правило "а") или следующему (правило "б") правилу.

б) ссылка появляется как "Symbol" в "SymbolsFromModule", "AssignedIdentifier" которого не содержит текстуально никаких "valuereference".

Примечание — Рекомендуется, чтобы идентификатор объекта был присвоен таким образом, чтобы можно было недвусмысленно ссылаться на модуль.

12.9 Глобальная ссылка на модуль "GlobalModuleReference" в "SymbolsFromModule" должна появляться в "ModuleDefinition" другого модуля, и, за исключением случая, когда он содержит непустой "DefinitiveIdentifier", "modulereference" может различаться в этих двух модулях.

Примечание — Отличающаяся "modulereference" в другом модуле должна использоваться, только когда импортируются символы с одним и тем же именем из двух разных модулей (модули были названы без учета 12.6). Использование альтернативных различных имен делает эти имена доступными для использования в теле модуля (см. 12.15).

12.10 Когда в ссылающемся модуле используются и "modulereference", и непустой "AssignedIdentifier", то последний должен рассматриваться как определяющий.

12.11 Когда указываемый модуль имеет непустой "DefinitiveIdentifier", тогда "GlobalModuleReference", указывающая этот модуль, не должна иметь пустой "AssignedIdentifier".

12.12 Когда в "Exports" выбрана альтернатива "SymbolsExported", тогда:

а) каждый "Symbol" в "SymbolsExported" должен удовлетворять в точности одному из следующих условий:

i) он является единственным, определенным в создаваемом модуле, или

ii) он появляется ровно один раз и только в альтернативе "SymbolsImported" в "Imports";

б) каждый "Symbol", для которого предназначена ссылка из внешнего модуля, должен быть включен в "SymbolsExported", и только эти "Symbol" могут быть указаны вне модуля, и

в) если нет таких "Symbol", то должна быть выбрана пустая альтернатива для "SymbolsExported" (а не для "Exports").

12.13 Когда выбрана альтернатива "empty" для "Exports", каждый "Symbol", определенный в модуле, может быть указан вне модуля.

Примечание — Альтернатива "empty" для "Exports" включена для совместимости с предшествующими версиями нотации.

12.14 Идентификаторы, которые появляются в "NamedNumberList", "Enumeration"или "NamedBitList", экспортируются неявно, если определяющая их ссылка на тип экспортируется или появляется как компонент (или подкомпонент) в экспортируемом типе.

12.15 Когда выбрана альтернатива "SymbolsImported" в "Imports", тогда:

а) каждый "Symbol" в "SymbolsFromModule" должен быть либо определен в теле модуля, либо представлен в разделе IMPORTS модуля, обозначенного "GlobalModuleReference" в "SymbolsFromModule". Импорт символа "Symbol", присутствующего в разделе IMPORTS указываемого модуля, допустим, если имеется только одно появление "Symbol" в этом разделе и "Symbol" не определен в указываемом модуле.

Примечание 1 — Не запрещается одно и то же имя символа, определенное в двух разных модулях, импортировать в другой модуль. Однако если одно и то же имя "Symbol" встречается несколько раз в разделе IMPORTS модуля А, то это имя "Symbol" не может быть экспортировано из А в другой модуль В;

б) если для "Exports" выбрана альтернатива "SymbolsExported" в определении модуля, обозначенного "GlobalModuleReference" в "SymbolsFromModule", то "Symbol" должен присутствовать в его "SymbolsExported";

в) только тот "Symbol" из присутствующих в "SymbolList" для "SymbolsFromModule" может быть использован в качестве символа в любой ссылке "External<X>Reference", который имеет ссылку "modulereference", обозначенную "GlobalModuleReference" в этом "SymbolsFromModule" (где <X> есть "value", "type", "object", "objectclass" или "objectset");

г) если нет такого "Symbol", то для "SymbolsImported" должна быть выбрана пустая альтернатива "empty".

Примечание 2 — Из в) и г) следует утверждение: "IMPORTS;" подразумевает, что модуль не может содержать "External<X>Reference";

д) все "SymbolsFromModule" в "SymbolsFromModuleList" должны содержать экземпляры ссылки "GlobalModuleReference", такие, что:

i) все "modulereference" в ней отличны друг от друга и от "modulereference", ассоциированной со ссылающимся модулем, и

ii) "AssignedIdentifier", когда он не пуст, обозначает значения идентификаторов объектов, которые отличны друг от друга и от значения идентификатора объекта (если он есть), ассоциированного со ссылающимся модулем.

12.16 Когда для "Imports" выбрана пустая альтернатива "empty", то модуль может ссылаться на "Symbol", определенный в другом модуле с помощью "External<X>Reference".

Примечание — Альтернатива "empty" для "Imports" включена для совместимости с предшествующими версиями нотации.

12.17 Идентификатор, который появляется в "NamedNumberList", "Enumeration" или "NamedBitList", импортируется неявно, если определяющая его ссылка на тип импортируется или появляется как компонент (или подкомпонент) в импортируемом типе.

12.17 bis Символ "Symbol" из "SymbolsFromModule" может появляться в "ModuleBody" в качестве ссылки "Reference". Смысловое значение, связанное с "Symbol", то же самое, какое он имеет в модуле, обозначенном соответствующей ссылкой "GlobalModuleReference".

12.18 Когда символ "Symbol" также появляется в "AssignmentList" (не рекомендуется) или в одном или нескольких других экземплярах "SymbolsFromModule", он должен использоваться только в ссылках "External<X>Reference". Если он таким образом не появляется, то он должен использоваться непосредственно как ссылка "Reference".

12.19 Различные альтернативы для присвоения "Assignment" определены в следующих разделах настоящего и последующих стандартов:

Альтернатива присвоения	Определяющий раздел
"TypeAssignment"	15.1
"ValueAssignment"	15.2
"ValueSetTypeAssignment"	15.4
"ObjectClassAssignment"	ГОСТ Р ИСО/МЭК 8824-2, 9.1
"ObjectAssignment"	ГОСТ Р ИСО/МЭК 8824-2, 11.1
"ObjectSetAssignment"	ГОСТ Р ИСО/МЭК 8824-2, 12.1
"ParameterizedAssignment"	ИСО/МЭК 8824-4, 8.1

Первый символ в любом "Assignment" является одной из альтернатив для "Reference", обозначающей определяемое ссылочное имя. Ни для каких двух присвоений в списке "AssignmentList" не может быть одинаковых ссылочных имен.

13 Ссылки на определения типов и значений

13.1 Последовательности, которые должны использоваться для ссылок на определения типов и значений, определяются следующими продукциями:

```
DefinedType : : =
        Externaltypereference|
        typereference    |
        ParameterizedType  |
        ParameterizedValueSetType
DefinedValue : : =
        Externalvaluereference     |
        valuereference   |
        ParameterizedValue
```

Тип, идентифицированный "ParameterizedType" и "ParameterizedValueSetType", и значение, идентифицированное "ParameterizedValue", определены в ИСО/МЭК 8824-4.

13.2 За исключением случаев, определенных в 12.18, альтернативы "typereference", "valuereference", "ParameterizedType", "ParameterizedValueSetType" или "ParameterizedValue" не должны использоваться, если ссылка не находится в том же теле "ModuleBody", в котором тип или значение были присвоены (см. 15.1 и 15.2) ссылке на тип или значение.

13.3 Внешние ссылки "Externaltypereference" и "Externalvaluereference" не должны использоваться, если соответствующей ссылке "typereference" или "valuereference":

а) не был присвоен, соответственно, тип или значение (см. 15.1 и 15.2), или

б) она не находится в разделе IMPORTS

в теле "ModuleBody", использованном для определения соответствующей ссылки "modulereference". Указание элементов в разделе IMPORTS другого модуля допустимо, только когда имеется не более одного экземпляра этого "Symbol" в данном разделе.

Примечание — Не запрещается один и тот же "Symbol", определенный в двух разных модулях, импортировать в другой модуль. Однако если один и тот же "Symbol" встречается несколько раз в разделе IMPORTS модуля А, то этот "Symbol" не может быть указан с использованием модуля А во внешней ссылке.

13.4 Внешняя ссылка должна использоваться в модуле только для указания элемента, определенного в другом модуле, и задается следующими продукциями:

```
Externaltypereference : : =
        modulereference
        "."
        typereference
Externalvaluereference : : =
        modulereference
        "."
        valuereference
```

Примечание — Дополнительные продукции внешних ссылок (ExternalClassReference, ExternalObjectReference и ExternalObjectSetReference) определены в ГОСТ Р ИСО/МЭК 8824-2.

13.5 Когда ссылающийся модуль определен с использованием альтернативы "SymbolsImported" для "Imports", "modulereference" во внешней ссылке должен появляться в "GlobalModuleReference" для ровно одного "SymbolsFromModule" в "SymbolsImported". Когда ссылающийся модуль определен с использованием пустой альтернативы "empty" для "Imports", "modulereference" во внешней ссылке должен появляться в "ModuleDefinition" для модуля (отличного от ссылающегося модуля), в котором определена ссылка "Reference".

13.6 Когда "DefinedType" используется как часть нотации, управляемой "Type" (например в "SubtypeElementSpec"), то "DefinedType" должен быть совместим с управляющим "Type", как определено в F.6.2.

13.7 Каждое появление "DefinedValue" в спецификации АСН.1 управляется "Type" и это "DefinedValue" должно указывать на значение типа, совместимого с управляющим "Type", как определено в F.6.2.

14 Нотация для обеспечения ссылок на компоненты АСН.1

14.1 Для многих целей требуются формальные ссылки на компоненты типов АСН.1, значений и пр. Одним из таких примеров является необходимость идентификации конкретного типа в некотором модуле АСН.1. В данном разделе определена нотация, которая может быть использована для обеспечения подобных ссылок.

14.2 Нотация позволяет идентифицировать любой компонент типов "множество" и "последовательность" (присутствующий в типе как обязательно, так факультативно).

14.3 На любую часть любого определения типа АСН.1 можно сослаться, используя синтаксическую конструкцию "AbsoluteReference":

```
AbsoluteReference : : =  "@" GlobalModuleReference
        "."
        ItemSpec
```

```
ItemSpec : : =
        typereference    |
        ItemId"." ComponentId
ItemId : : = ItemSpec
ComponentId : : =
      identifier | number | "*"
```

Примечание — Продукция для "AbsoluteReference" не используется в настоящем стандарте. Она введена для целей, указанных в 14.1.

14.4 Ссылка "GlobalModuleReference" идентифицирует модуль АСН.1 (см. 12.1).

14.5 Ссылка "typereference" идентифицирует любой тип АСН.1, определенный в модуле, идентифицированном "GlobalModuleReference".

14.6 "ComponentId" в каждой "ItemSpec" идентифицирует компонент типа, который идентифицирован "ItemId". Последним должен быть "ComponentId", если идентифицируемый им компонент не является типом "множество", "последовательность", "множество-из", "последовательность-из" или выборочным типом.

14.7 Альтернатива "identifier" для "ComponentId" может быть использована, если порождающий "ItemId" является типом "множество" или "последовательность" и обязательно должен быть одним из "identifier" для "NamedType" в "ComponentTypeList" этого множества или последовательности. Она также может использоваться, если "ItemId" идентифицирует выборочный тип, и тогда он обязательно должен быть одним из "identifier" для "NamedType" в "AlternativeTypeList" этого выборочного типа. При иных обстоятельствах эта альтернатива использоваться не может.

14.8 Альтернатива "number" для "ComponentId" может быть использована, только если "ItemId" является типом "множество-из" или "последовательность-из". Значение числа "number" идентифицирует экземпляр типа в множестве-из или последовательности-из, при этом значение "1" идентифицирует первый экземпляр типа. Значение "0" идентифицирует концептуальный компонент целого типа (не присутствующий явно при передаче и называемый "счетчик итераций"), который содержит число экземпляров типа в множестве-из или последовательности-из, имеющихся в значении охватывающего типа.

14.9 Альтернатива "*" для "ComponentId" может быть использована, только если "ItemId" является типом "множество-из" или "последовательность-из". Любая семантика, связанная с использованием "*" для "ComponentId", применяется ко всем компонентам множества-из и последовательности-из.

Примечание — В следующем примере:

```
M DEFINITIONS : : =  BEGIN
    T  : : =  SEQUENCE {
        a BOOLEAN,
        b SET OF INTEGER
    }
END
```

компонент "T" может быть указан в тексте вне модуля АСН.1 (или в комментарии) следующим образом:

если (@M.T.b.0 нечетное), то
(@M.T.b.* должно быть нечетным),

использованным для утверждения, что если количество компонентов в "b" нечетное, то все компоненты "b" должны быть нечетными.

15 Присвоение типов и значений

15.1 Ссылке на тип "typereference" должен быть присвоен тип нотацией, заданной продукцией "TypeAssignment":

```
TypeAssignment  : : =
        typereference
        ": : ="
        Type
```

Ссылка "typereference" не должна быть зарезервированным словом АСН.1 (см. 11.18).

15.2 Ссылке на значение "valuereference" должно быть присвоено значение нотацией, заданной продукцией "ValueAssignment":

```
ValueAssignment  : : =
        valuereference
        Type
        ": : ="
        Value
```

Значение "Value", которое присваивается ссылке "valuereference", управляется "Type" и должно быть в нотации для значения типа, определяемого "Type" (как сказано в 15.3).

15.3 Значение "Value" является нотацией для значения типа, если:

а) либо "Value" есть нотация "BuiltinValue" для типа (см. 16.8),

б) либо "Value" есть нотация "DefinedValue" для значения этого типа.

15.4 Ссылке на тип "typereference" может быть присвоено множество значений нотацией, заданной продукцией "ValueSetTypeAssignment":

```
ValueSetTypeAssignment : : =  typereference
                              Type
                              ": : ="
                              ValueSet
```

Эта нотация присваивает ссылке "typereference" тип, определяемый как подтип типа, обозначенного "Type", содержащий только те значения, которые специфицированы или допускаются "ValueSet". Ссылка "typereference" не должна быть зарезервированным словом АСН.1 (см. 11.18) и может указываться как тип. "ValueSet" определяется в 15.5.

15.5 Множество значений некоторого типа должно быть специфицировано нотацией "ValueSet":

```
ValueSet  : : =  "{" ElementSetSpec  "}"
```

Множество значений включает в себя все значения, которые, по крайней мере один раз, специфицированы "ElementSetSpec" (см. раздел 46).

16 Определение типов и значений

16.1 Тип специфицируется нотацией "Type":

```
Type : : = BuiltinType | ReferencedType | ConstrainedType
```

16.2 Встроенные типы АСН.1 специфицируются нотацией "BuiltinType", определяемой следующим образом:

```
BuiltinType : : =
    BitStringType                |
    BooleanType                  |
    CharacterStringType          |
    ChoiceType                   |
    EmbeddedPDVType              |
    EnumeratedType               |
    ExternalType                 |
    InstanceOfType               |
    IntegerType                  |
    NullType                     |
    ObjectClassFieldType         |
    ObjectIdentifierType         |
    OctetStringType              |
    RealType                     |
    SequenceType                 |
    SequenceOfType               |
    SetType                      |
    SetOfType                    |
    TaggedType
```

Различные нотации "BuiltinType" определяются в следующих разделах настоящего стандарта (если не оговорено иное):

```
BitStringType          21
BooleanType            17
CharacterStringType    35
```

ChoiceType	28
EmbeddedPDVType	32
EnumeratedType	19
ExternalType	33
InstanceOfType	ГОСТ Р ИСО/МЭК 8824-2, приложение С
IntegerType	18
NullType	23
ObjectClassFieldType	ГОСТ Р ИСО/МЭК 8824-2, 14.1
ObjectIdentifierType	31
OctetStringType	22
RealType	20
SequenceType	24
SequenceOfType	25
SetType	26
SetOfType	27
TaggedType	30

16.3 Указываемые типы АСН.1 специфицируются нотацией "ReferencedType":

```
ReferencedType : : =
        DefinedType                    |
        UsefulType                     |
        SelectionType                  |
        TypeFromObject                 |
        ValueSetFromObjects
```

Нотация "ReferencedType" обеспечивает альтернативный способ указания некоторого другого (а в конечном счете – встроенного) типа. Различные нотации "ReferencedType" и методы, которыми указываемый ими тип определяется, специфицированы в следующих пунктах и разделах настоящего стандарта (если не оговорено иное):

DefinedType	13.1
UsefulType	40.1
SelectionType	29
TypeFromObject	ГОСТ Р ИСО/МЭК 8824-2, раздел 15
ValueSetFromObjects	ГОСТ Р ИСО/МЭК 8824-2, раздел 15

16.4 "ConstrainedType" определен в разделе 44.

16.5 Настоящим стандартом требуется использование нотации "NamedType" в спецификации компонентов типов "множество", "последовательность" и выборочных типов. Нотация для "NamedType" следующая:

```
NamedType : : =  identifier Type
```

16.6 Идентификатор "identifier" используется для недвусмысленной ссылки на тип "множество", "последовательность" и выборочного типа в нотации значения и в ограничениях связей компонентов (см. ИСО/МЭК 8824-3). Он не является частью типа и не влияет на него.

16.7 Значение некоторого типа должно быть специфицировано нотацией "Value":

```
Value : : =  BuiltinValue | ReferencedValue | ObjectClassFieldValue
```

Примечание — "ObjectClassFieldValue" определяется в ГОСТ Р ИСО/МЭК 8824-2, 14.6.

16.8 Значения встроенных типов АСН.1 могут быть специфицированы нотацией "BuiltinValue", определенной следующим образом:

```
BuiltinValue : : =
        BitStringValue          |
        BooleanValue            |
        CharacterStringValue    |
        ChoiceValue             |
        EmbeddedPDVValue        |
        EnumeratedValue         |
        ExternalValue           |
        InstanceOfValue         |
        IntegerValue            |
```

```
        NullValue                |
        ObjectIdentifierValue    |
        OctetStringValue         |
        RealValue                |
        SequenceValue            |
        SequenceOfValue          |
        SetValue                 |
        SetOfValue               |
        TaggedValue
```

Различные нотации "BuiltinValue" определены в тех же самых разделах, что и соответствующие нотации "BuiltinType" (см. 16.2).

16.9 Указываемые значения АСН.1 специфицируются нотацией "ReferencedValue":

```
ReferencedValue : : =
    DefinedValue     |
    ValueFromObject
```

Нотация "ReferencedValue" обеспечивает альтернативный способ указания некоторого другого (а в конечном счете — встроенного) значения. Различные нотации "ReferencedValue" и методы, которыми указываемое ими значение определяется, специфицированы в следующих пунктах и разделах настоящего стандарта (если не оговорено иное):

DefinedValue	13.1
ValueFromObject	ГОСТ Р ИСО/МЭК 8824-2, раздел 15

16.10 Независимо от того, является ли тип "BuiltinType", "ReferencedType"или "ConstrainedType", его значение может быть задано либо "BuiltinValue", либо "ReferencedValue" этого типа.

16.11 Значение типа, указанного с использованием нотации "NamedType", должно быть определено нотацией "NamedValue":

```
NamedValue : : =  identifier Value
```

где "identifier" — тот же самый, который был использован в нотации "NamedType".

Примечание — "identifier" является частью нотации, но не образует часть самого значения. Он используется для недвусмысленной ссылки на компоненты типов "множество", "последовательность" и выборочного типа.

16.12 Подразумеваемое или явное присутствие маркера расширения в определении типа не влияет на значение нотации. Значение нотации для типа с маркером расширения точно такое же, как если бы маркер отсутствовал.

17 Нотация для булевского типа

17.1 Булевский тип (см. 3.8.7) должен указываться нотацией "BooleanType":

```
BooleanType : : =  BOOLEAN
```

17.2 Типы, определенные с этой нотацией, имеют тег универсального класса 1.

17.3 Значение булевского типа (см. 3.8.66 и 3.8.88) должно определяться нотацией "BooleanValue":

```
BooleanValue : : =  TRUE | FALSE
```

18 Нотация для целочисленного типа

18.1 Целочисленный тип (см. 3.8.40) должен указываться нотацией "IntegerType":

```
IntegerType : : =
                INTEGER                  |
                INTEGER "{" NamedNumberList "}"
NamedNumberList : : =
                NamedNumber              |
                NamedNumberList "," NamedNumber
NamedNumber : : =
                identifier "(" SignedNumber ")"    |
                identifier "(" DefinedValue ")"
SignedNumber : : =  number | "–" number
```

18.2 Вторая альтернатива для "SignedNumber" не должна использоваться, если "number" есть нуль.

18.3 "NamedNumberList" не является существенной составной частью определения типа. Эта последовательность используется только в нотации для значения, определенной в 18.9.

18.4 Ссылка "valuereference" в "DefinedValue" должна быть целочисленного типа.

Примечание — Так как "identifier" не может использоваться для спецификации значения, связанного с "NamedNumber", то "DefinedValue" никогда не может быть ошибочно интерпретировано как "IntegerValue". Следовательно, в следующем случае

```
a INTEGER : : = 1
T1 : : = INTEGER{a(2)}
T2 : : = INTEGER{a(3), b(a)}
c T2 : : = b
d T2 : : = a
```

"c" обозначает значение 1, так как не может относится ни ко второму, ни к третьему появлению "a" в этом примере, а "d" обозначает значение 3.

18.5 Значения каждого из "SignedNumber" и "DefinedValue", входящего в "NamedNumberList", должны быть различными и представлять разные значения целочисленного типа.

18.6 Все "identifier" входящие в "NamedNumberList", должны быть различными.

18.7 Порядок последовательностей "NamedNumber" в "NamedNumberList" не существенен.

18.8 Типы, определенные этой нотацией, имеют тег универсального класса 2.

18.9 Значение целочисленного типа должно определяться нотацией "IntegerValue":

```
IntegerValue : : =
        SignedNumber   |
        identifier
```

18.10 Идентификатор "identifier" в продукции "IntegerValue" должен быть одним из идентификаторов "identifier" в "IntegerType", с которым связано значение, и представляет соответствующее число.

Примечание — Когда указывается целочисленное значение, для которого был определен идентификатор "identifier", использование формы "identifier" для "IntegerValue" является предпочтительным.

19 Нотация для перечислимого типа

19.1 Перечислимый тип (см. 3.8.24) должен указываться нотацией "EnumeratedType":

```
EnumeratedType : : =
        ENUMERATED "{" Enumerations "}"
Enumerations : : = RootEnumeration    |
        RootEnumeration "," "..."          |
        RootEnumeration "," "..." "," AdditionalEnumeration
RootEnumeration : : = Enumeration
AdditionalEnumeration : : = Enumeration
Enumeration : : =
        EnumerationItem | EnumerationItem "," Enumeration
EnumerationItem : : =
        identifier | NamedNumber
```

Примечания

1 Каждое значение "EnumeratedType" имеет идентификатор, который связан с отличным от других целым числом. Однако не подразумевается, что само значение должно иметь какую-либо целочисленную семантику. Спецификация альтернативы "NamedNumber" для "EnumerationItem" обеспечивает управление представлением значения для облегчения совместимости выражений.

2 Численные значения внутри "NamedNumber" в "RootEnumeration" не обязательно должны быть упорядоченными или последовательными, а численные значения внутри "NamedNumber" в "AdditionalEnumeration" должны быть упорядоченными или, но не обязательно, последовательными.

19.2 Для каждой альтернативы "NamedNumber" идентификаторы "identifier" и "SignedNumber" должны отличаться от всех других "identifier" и "SignedNumber" в "Enumeration". Кроме того, для каждой "NamedNumber" выполняются положения 18.2 и 18.4.

19.3 Всем элементам перечисления "EnumerationItem" (в "EnumeratedType"), являющимся идентификаторами "identifier", последовательно присваиваются неотрицательные целые. Для этого последовательность целых начинается с 0, но из нее исключаются те, которые уже задействованы в элементах перечисления "EnumerationItem", являясь "NamedNumber".

Примечание — Целочисленные значения связываются с "EnumerationItem" для определения правил кодирования. В противном случае они не используются в спецификации АСН.1.

19.4 Значение каждого нового "EnumerationItem" должно быть больше, чем все ранее определенные "AdditionalEnumeration" в типе.

19.5 Когда "NamedNumber" используется в определяющем "EnumerationItem" в "AdditionalEnumeration", связанное с ним значение должно отличаться от значений всех ранее определенных "EnumerationItem" (в этом типе) независимо от того, встречаются ранее определенные "EnumerationItem" в корне перечисления или нет. Например:

```
A : : = ENUMERATED {a, b, ..., c(0)} - - недопустимо, т. к. 'a' и 'c' равны 0
B : : = ENUMERATED {a, b, ..., c, d(2)} - - недопустимо, т. к. 'c' и 'd' равны 2
C : : = ENUMERATED {a, b(3), ..., c(1)}      - - допустимо, c = 1
D . : = ENUMERATED {a, b, ..., c(2)}         - - допустимо, c = 2
```

19.6 Значение, связанное с первой альтернативой "EnumerationItem" в "AdditionalEnumeration", которая есть "identifier" (а не "NamedNumber"), должно быть наименьшим значением, для которого "EnumerationItem" не определен в "RootEnumeration", и все предшествующие "EnumerationItem" в "AdditionalEnumeration" (если они есть) должны быть меньше. Например являются допустимыми следующие записи:

```
A : : = ENUMERATED {a, b, ..., c}          - - c = 2
B : : = ENUMERATED {a, b, c(0), ...,  d}   - - d = 3
C : : = ENUMERATED {a, b, ..., c(3),  d}   - - d = 4
D : : = ENUMERATED {a, z(25), ..., d}      - -  d = 1
```

19.7 Перечислимый тип имеет тег универсального класса 10.

19.8 Значение перечислимого типа должно определяться нотацией "EnumeratedValue":

```
EnumeratedValue : : = identifier
```

19.9 Идентификатор "identifier" в перечислимом значении "EnumeratedValue" должен быть равен значению "identifier" в последовательности "EnumeratedType", с которым оно связывается.

20 Нотация для действительного типа

20.1 Действительный тип (см. 3.8.52) должен указываться нотацией "RealType":

```
RealType : : = REAL
```

20.2 Действительный тип имеет тег универсального класса 9.

20.3 Значениями действительного типа являются значения PLUS-INFINITY, MINUS-INFINITY и действительные числа, задаваемые следующей формулой, содержащей три целых числа — M, B и E:

$$M \cdot B^E,$$

где M — мантисса, B — основание, а E — показатель степени.

20.4 Действительный тип имеет ассоциированный тип, который используется для задания точности абстрактных значений действительного типа и обеспечения нотаций значений и подтипов действительного типа.

Примечание — Правилами кодирования может определяться другой тип, который используется для спецификации кодирования, или может быть задано кодирование без ссылки на ассоциированный тип. В частности, кодирование в BER и PER обеспечивает двоично-кодированное десятичное (BCD) кодирование, если основание равно 10, и кодирование, допускающее эффективное преобразование в машинное представление с плавающей точкой и обратно, если основание равно 2.

20.5 Ассоциированный тип для определения значения и подтипов есть (с обязательными комментариями):

```
SEQUENCE {
    mantissa INTEGER,
```

```
        base INTEGER (2|10),
        exponent INTEGER
          - - Ассоциированное математическое действительное число
          - - есть "mantissa", умноженная на "base" в степени "exponent"
    }
```

Примечания

1 Значения, представленные по основанию 2 и 10, рассматриваются как различные абстрактные значения, даже если их вычисление дает одно и то же действительное число, и могут иметь разную прикладную семантику.

2 Нотация "REAL (WITH COMPONENTS { ..., base(10)})" может быть использована для ограничения множества значений по абстрактному основанию 10 (и, аналогично, — основанию 2).

3 Этот тип может предоставлять точное, конечное значение любого числа, которое может храниться в памяти машины с плавающей точкой, и любого числа с конечным символьно-десятичным представлением.

20.6 Значение действительного типа должно определяться нотацией "RealValue":

```
RealValue : : =
        NumericRealValue | SpecialRealValue
NumericRealValue : : = 0
        SequenceValue - - Значение ассоциированного типа
SpecialRealValue : : =
        PLUS-INFINITY | MINUS-INFINITY
```

Форма "0" должна использоваться для нулевых значений, альтернативная форма "NumericRealValue" для нулевых значений использоваться не может.

21 Нотация для типа "битовая строка"

21.1 Тип "битовая строка" (см. 3.8.6) должен указываться нотацией "BitStringType":

```
BitStringType : : =
        BIT STRING
        BIT STRING "{" NamedBitList "}"
NamedBitList : : =
        NamedBit                |
        NamedBitList "," NamedBit
NamedBit : : =
        identifier "(" number ")"   |
        identifier "(" DefinedValue ")"
```

21.2 Первый бит в битовой строке называется нулевым. Последний бит в битовой строке называется завершающим.

Примечание — Эта терминология используется в спецификации значения нотации и определении правил кодирования.

21.3 "DefinedValue" должно быть ссылкой на неотрицательное значение целочисленного типа.

21.4 Значения последовательностей "number" и "DefinedValue", появляющихся в списке поименованных битов "NamedBitList", должны быть попарно различными, и являются номерами различных битов в значении битовой строки.

21.5 Все идентификаторы "identifier", появляющиеся в "NamedBitList", должны быть различными.

Примечания

1 Порядок последовательностей продукций "NamedBit" в "NamedBitList" не существенен.

2 Так как "identifier", который появился в "NamedBitList", не может быть использован для спецификации значения, связанного с "NamedBit", то "DefinedValue" никогда не может быть ошибочно интерпретировано как "IntegerValue". Следовательно, в следующем случае:

```
a INTEGER : : = 1
T1 : : = INTEGER{a(2)}
T2 : : = BIT STRING {a(3), b(a)}
```

последнее появление "a" обозначает значение 1, которое не может быть указано ни при втором, ни при третьем появлении "a".

21.6 Присутствие "NamedBitList" не влияет на множество абстрактных значений этого типа. Значения, содержащие биты 1, отличные от поименованных битов, допускаются.

21.7 Когда список поименованных битов "NamedBitList" используется в определяющем типе битовой строки, правила кодирования АСН.1 могут добавлять (или убирать) произвольное число завершающих битов 0 к (или из) значениям(ий), которые кодируются или декодируются. Следовательно, проектировщики реализаций должны гарантировать, что различные семантики не связаны со значениями, различающимися только числом завершающих битов 0.

21.8 Этот тип имеет тег универсального класса 3.

21.9 Значение типа "битовая строка" должно определяться нотацией "BitStringValue":

```
BitStringValue : : =
        bstring                         |
        hstring                         |
        "{" IdentifierList "}"          |
        "{" "}"
IdentifierList : : =
        identifier          |
        IdentifierList "," identifier
```

21.10 Каждый из идентификаторов "identifier" в значении "BitStringValue" должен быть тем же самым, что и "identifier" в последовательности продукций "BitStringType", с которой связывается значение.

21.11 Нотация "BitStringValue" обозначает значение битовой строки с единицами в позициях, заданных номерами, соответствующими идентификаторам "identifier", и с нулями – в остальных позициях.

Примечание — Последовательность продукций "{" "}" используется для обозначения битовой строки, не содержащей ни одного бита.

21.12 При спецификации правил кодирования для битовой строки будут использованы термины "первый бит" и "завершающий бит", где первый бит есть нулевой бит (см. 21.2).

21.13 При использовании нотации "bstring" первый бит записывается слева, а завершающий бит — справа.

21.14 При использовании нотации "hstring" старший бит каждой шестнадцатеричной цифры соответствует левому биту в битовой строке.

Примечание — Эта нотация никоим образом не ограничивает способ размещения правилами кодирования битовой строки по октетам для передачи.

21.15 Нотация "hstring" может использоваться, если только значение битовой строки состоит из количества бит, кратного четырем.

Пример

```
'A98A'H
```

и

```
'1010100110001010'B
```

являются альтернативными обозначениями одного и того же значения битовой строки. Если тип был определен с использованием списка поименованных битов "NamedBitList", то (единственный) завершающий нуль не является частью значения, которое, таким образом, имеет длину 15 бит. Если тип был определен без "NamedBitList", то завершающий нуль является частью значения, которое имеет длину 16 бит.

22 Нотация для типа "строка октетов"

22.1 Тип "строка октетов" (см. 3.8.48) должен указываться нотацией "OctetStringType":

```
OctetStringType : : =  OCTET STRING
```

22.2 Этот тип имеет тег универсального класса 4.

22.3 Значение типа "строка октетов" должно определяться нотацией "OctetStringValue":

```
OctetStringValue : : =
        bstring    |
        hstring
```

22.4 Для строк октетов при спецификации правил кодирования октеты указываются терминами "первый октет" и "завершающий октет", а для битов внутри октета — терминами "старший бит" и "младший бит".

22.5 При использовании нотации "bstring" самый левый бит является старшим битом первого октета. Если "bstring" состоит из количества бит, не кратного восьми, она должна интерпретироваться таким образом, как если бы она содержала дополнительные завершающие нули, которые сделают ее длину наименьшей из кратных восьми.

22.6 При использовании нотации "hstring" крайняя левая шестнадцатеричная цифра должна быть старшим полуоктетом первого октета.

22.7 Если "hstring" состоит из нечетного числа шестнадцатеричных цифр, то она должна интерпретироваться таким образом, как если бы она содержала еще одну дополнительную завершающую нулевую шестнадцатеричную цифру.

23 Нотация для вырожденного типа

23.1 Вырожденный тип (см. 3.8.43) должен указываться нотацией "NullType":

```
NullType : : =  NULL
```

23.2 Этот тип имеет тег универсального класса 5.

23.3 Значение вырожденного типа должно указываться нотацией "NullValue":

```
NullValue : : =  NULL
```

24 Нотация для типов "последовательность"

24.1 Нотацией для определения типа "последовательность" (см. 3.8.56) должна быть "SequenceType":

```
SequenceType : : =  SEQUENCE "{" "}"  |
    SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"          |
    SEQUENCE "{" ComponentTypeLists "}"
ExtensionAndException : : =  "..." | "..." ExceptionSpec
OptionalExtensionMarker : : =  "," "..." | empty
ComponentTypeLists : : =  RootComponentTypeList |
    RootComponentTypeList "," ExtensionAndException ExtensionAdditions OptionalExtensionMarker |
    RootComponentTypeList "," ExtensionAndException ExtensionAdditions
                              ExtensionEndMarker "," RootComponentTypeList      |
    ExtensionAndException ExtensionAdditions ExtensionEndMarker "," RootComponentTypeList
RootComponentTypeList : : = ComponentTypeList
ExtensionEndMarker : : = "," "..."
ExtensionAdditions : : = "," ExtensionAdditionList | empty
ExtensionAdditionList : : = ExtensionAddition | ExtensionAdditionList "," ExtensionAddition
ExtensionAddition : : = ComponentType | ExtensionAdditionGroup
ExtensionAdditionGroup : : = "[[" ComponentTypeList "]]"
ComponentTypeList : : =
    ComponentType                           |
    ComponentTypeList "," ComponentType
ComponentType : : =
    NamedType                               |
    NamedType OPTIONAL                      |
    NamedType DEFAULT Value                 |
    COMPONENTS OF Type
```

24.2 Когда продукция "ComponentTypeLists" встречается в определении модуля, для которого выбрано автоматическое тегирование (см. 12.3), и никакой из экземпляров "NamedType" в любой из

первых трех альтернатив для "ComponentType" не содержит "TaggedType", то преобразование автоматического тегирования выбрано для всего "ComponentTypeLists"; в противном случае — нет.

Примечания

1 Использование нотации "TaggedType" в определении списка компонентов для типа "последовательность" позволяет спецификатору управлять тегами в противоположность автоматическому присваиванию в методе автоматического тегирования. Следовательно, в следующем случае:

T : : = SEQUENCE { a INTEGER, b [1] BOOLEAN, c OCTET STRING }

автоматическое тегирование не применяется к списку компонентов a, b, c, даже если это определение типа "последовательность" T встретилось в модуле, для которого было выбрано автоматическое тегирование.

2 Только экземпляры продукции "ComponentTypeList", встретившиеся в модуле, для которого было выбрано автоматическое тегирование, являются кандидатами для преобразования автоматического тегирования.

24.3 Вопрос о применении преобразования автоматического тегирования решается индивидуально для каждого экземпляра списка "ComponentTypeLists" и до преобразования COMPONENTS OF, определенного в 24.4. Однако, как определено в 24.7—24.9, преобразование автоматического тегирования (если оно применяется) производится после преобразования COMPONENTS OF.

Примечание — В результате применение автоматических тегов подавляется тегами, явно присутствующими в "ComponentTypeLists", но не тегами, присутствующими в "Type", следующем за COMPONENTS OF.

24.4 Тип "Type" в нотации "COMPONENTS OF Type" должен быть типом "последовательность". Нотация "COMPONENTS OF Type" должна использоваться для определения включения, в этом месте списка компонентов, всех компонентов указанного типа, за исключением любых маркеров расширения и расширяющих дополнений, которые могут присутствовать в "Type". (В "COMPONENTS OF Type" включается только "RootComponentTypeList" типа "Type"; маркеры расширения и расширяющие дополнения, если они есть, игнорируются нотацией "COMPONENTS OF Type"). При этом преобразовании игнорируются все ограничения, применяемые к указываемому типу.

Примечание — Это преобразование логически завершается до удовлетворения требований последующих подразделов.

24.5 В каждом из последующих подразделов идентифицированы серии появлений "ComponentType" либо в корне, либо в расширяющих дополнениях, либо и в том, и в другом. Правило 24.5.1 должно применяться для всех таких серий.

24.5.1 Для каждой серии из одного или нескольких последовательных появлений типов компонентов "ComponentType", отмеченных как OPTIONAL или DEFAULT, теги этих "ComponentType" и любого непосредственно следующего типа компонента в серии должны быть различными (см. раздел 30). Если было выбрано автоматическое тегирование, то требование, чтобы теги были различными, применяется только после осуществления автоматического тегирования, и всегда должно быть удовлетворено, если применялось автоматическое тегирование.

24.5.2 Правило 24.5.1 должно применяться к сериям "ComponentType" в корне.

24.5.3 Правило 24.5.1 должно применяться к завершенным сериям "ComponentType" в корне или в расширяющих дополнениях в текстуальном порядке их появления в определении типа (игнорируя все скобки версий и многоточия).

24.6 Когда используется третья или четвертая альтернатива "ComponentTypeLists", все "ComponentType" в расширяющих дополнениях должны иметь теги, отличные от тегов текстуально последующих "ComponentType" до первого "ComponentType" который не отмечен как OPTIONAL или DEFAULT в завершающем "RootComponentTypeList" включительно (если такой есть).

24.7 Преобразование автоматического тегирования появления списка "ComponentTypeLists" логически осуществляется после преобразования, определенного в 24.4, но только если 24.2 определяет, что оно должно применяться к этому экземпляру "ComponentTypeLists". Преобразование автоматического тегирования изменяет каждый тип компонента "ComponentType" в "ComponentTypeLists", заменяя исходный тип "Type" в продукции "NamedType" замещением "TaggedType", определенным в 24.9.

24.8 Если действует автоматическое тегирование и "ComponentType" в корне расширения не имеют тегов, то "ComponentType" в "ExtensionAdditionList" должен быть тегированным типом.

24.9 Замещающий тегированный тип "TaggedType" определяется следующим образом:

а) в замещающей нотации "TaggedType" используется альтернатива "Tag Type";

б) "Class" замещения "TaggedType" пуст (т. е. тегирование контекстно зависимое);

в) "ClassNumber" в замещении "TaggedType" есть нулевое значение тега для первого "ComponentType" в "RootComponentTypeList" или для первого "NamedType" в "AlternativeTypeLists", единица — для второго и т. д. по мере возрастания номеров тегов;

г) "ClassNumber" в замещении "TaggedType" первого "ComponentType" в "ExtensionAdditionList" есть нуль, если "RootComponentTypeList" пропущен, в противном случае он на единицу больше самого большого "ClassNumber" в "RootComponentTypeList"; следующий "ComponentType" в "ExtensionAdditionList" имеет "ClassNumber" на единицу больше первого и т. д. по мере возрастания номеров тегов;

д) "Type" в замещении "TaggedType" есть исходный тип "Type", который будет замещен.

Примечания

1 Правила, управляющие спецификацией явного или неявного тегирования для замещения "TaggedType", приведены в 30.6. Автоматическое тегирование всегда неявное, если только "Type" не является выборочным типом, нотацией открытого типа или пустой ссылкой "DummyReference" (см. ИСО/МЭК 8824-4, 8.3), когда тегирование является явным.

2 Если удовлетворены требования 24.7, то теги компонентов полностью определены и не изменяются, даже когда тип "последовательность" указывается в определении компонента в другом списке "ComponentTypeList", для которого применяется автоматическое тегирование. Таким образом, в следующем случае:

```
T : : =  SEQUENCE { a Ta, b Tb, c Tc }
E : : =  SEQUENCE { f1 E1, f2 T, f3 E3 }
```

теги, присоединенные к a, b, c не изменятся возможным автоматическим тегированием, примененным к компонентам E.

3 Когда тип "последовательность" встречается как "Type" в "COMPONENTS OF Type", каждое появление в нем "ComponentType" дублируется благодаря применению 24.4 до возможного применения автоматического тегирования к ссылающемуся типу "последовательность". Таким образом, в следующем случае:

```
T : : =  SEQUENCE { a Ta, b SEQUENCE  { b1 T1, b2 T2,  b3 T3 }, c Tc }
W : : =  SEQUENCE { x Wx, COMPONENTS OF T, y Wy }
```

теги элементов a, b, c в последовательности T не обязательно должны быть теми же самыми, что и теги элементов a, b, c в последовательности W, если W была определена в окружении автоматического тегирования, но теги элементов b1, b2 и b3 — одни и те же в последовательностях T и W. Другими словами, преобразование автоматического тегирования применяется только один раз к данному списку "ComponentTypeLists".

4 Введение подтипа не влияет на автоматическое тегирование.

5 Когда имеет место автоматическое тегирование, вставка новых компонентов может привести к изменениям других компонентов, вызванным побочным эффектом модификации тегов.

24.10 Если используется ключевое слово "OPTIONAL" или "DEFAULT", то соответствующее значение может быть опущено из значения нового типа.

24.11 Если встречает ключевое слово "DEFAULT", то отсутствие значения этого типа должно быть в точности эквивалентно вставке значения, определенного "Value", которое, в свою очередь, должно быть нотацией для значения типа, определенного "Type" в последовательности продукций "NamedType".

24.12 Значение, соответствующее "ExtensionAdditionGroup" (всем компонентам вместе), является факультативным. Однако если такое значение присутствует, то должно присутствовать и значение, соответствующее компонентам в заключенном в скобки "ComponentTypeList", которые не отмечены как OPTIONAL или DEFAULT.

24.13 Идентификаторы "identifier" во всех последовательностях продукций "NamedType" списка "ComponentTypeLists" (вместе с теми, которые получены раскрытием COMPONENTS OF) должны быть различными.

24.14 Значение для данного типа, расширяющего дополнения, не должно устанавливаться, если не установлены значения для всех типов расширяющего дополнения, не отмеченных как OPTIONAL или DEFAULT, которые логически находятся между этим типом расширяющего дополнения и корнем расширения.

Примечания:

1 Когда тип наращивается от корня расширения (версия 1) через версию 2 к версии 3 путем добавления новых расширяющих дополнений, присутствие в кодировании любых дополнений из версии 3 требует присутствия в кодировании всех дополнений версии 2, не отмеченных как OPTIONAL или DEFAULT.

2 "ComponentType", которые имеются в расширяющих дополнениях, но не содержатся в "ExtensionAdditionGroup", должны кодироваться всегда, если они не отмечены как OPTIONAL или DEFAULT, за исключением

значения данных уровня представления, переданных отправителем, который использует предшествующую версию абстрактного синтаксиса, где не определена продукция "ComponentType".

3 Рекомендуется использовать продукцию "ExtensionAdditionGroup", так как:

а) она может привести к более компактному кодированию в зависимости от правил кодирования (например, PER);

б) синтаксис более точен в том отношении, если он ясно указывает, что значение типа, определенного в "ExtensionAdditionList" и не отмеченного как OPTIONAL или DEFAULT, всегда должно присутствовать в кодировании, если кодируется группа расширяющих дополнений, в которой определен тип (см. примечание 1);

в) синтаксис ясно указывает, какие типы в "ExtensionAdditionList" должны поддерживаться приложением как группа.

24.15 Все типы "последовательность" имеют тег универсального класса 16.

Примечание — Типы "последовательность" (см. 25.2).

24.16 Значение типа "последовательность" должно определяться нотацией

```
SequenceValue : : =
        "{" ComponentValueList "}" |
        "{"  "}"
ComponentValueList : : =
        NamedValue                |
        ComponentValueList ","  NamedValue
```

24.17 Нотация "{" "}" должна использоваться, только если:

а) все последовательности "ComponentType" в "SequenceType" помечены как "DEFAULT", или "OPTIONAL", а все значения — опущены, или

б) нотацией типа была "SEQUENCE {}".

24.18 Должно быть по одному значению "NamedValue" для каждого типа "NamedType" в "SequenceType", который не отмечен как "DEFAULT" или "OPTIONAL", и эти значения должны быть в том же самом порядке, что и соответствующие последовательности "NamedType".

25 Нотация для типов "последовательность-из"

25.1 Нотацией для определения типа "последовательность-из" (см. 3.8.57) из другого типа должна быть "SequenceOfType":

```
SequenceOfType : : =  SEQUENCE OF Type
```

25.2 Все типы "последовательность-из" имеют тег универсального класса 16.

Примечание — Типы "последовательность" имеют тот же самый тег, что и типы "последовательность-из" (см. 24.15).

25.3 Значение типа "последовательность-из" должно определяться нотацией "SequenceOfValue":

```
SequenceOfValue : : =  "{" ValueList  "}"  | "{"  "}"
ValueList : : =
        Value   |
        ValueList  ","  Value
```

Нотация "{" "}" используется, когда SequenceOfValue есть пустой список.

25.4 Каждое значение "Value" в списке значений "ValueList" должно быть нотацией для значения типа, заданного в "SequenceOfType".

Примечание — Семантическое значение может быть связано с порядком этих значений.

26 Нотация для типов "множество"

26.1 Нотацией для определения типа "множество" (см. 3.8.58) из других типов должна быть "SetType":

```
SetType : : =  SET  "{"  "}"        |
               SET "{" ExtensionAndException OptionalExtensionMarker  "}" |
               SET "{" ComponentTypeLists  "}"
```

Продукции "ComponentTypeLists", "ExtensionAndException" и "OptionalExtensionMarker" определены в 24.1.

26.2 "Type" в нотации "COMPONENTS OF Type" должен быть типом "множество". Нотация "COMPONENTS OF Type" должна использоваться для определения включения, в данном месте в списке компонентов, всех типов компонентов, появляющихся в указанном типе, за исключением любых маркеров расширения и расширяющих дополнений, которые могут присутствовать в "Type". (В "COMPONENTS OF Type" включается только "RootComponentTypeList" типа "Type"; маркеры расширения и расширяющие дополнения, если они есть, игнорируются нотацией "COMPONENTS OF Type".) При этом преобразовании игнорируются все ограничения, применяемые к указываемому типу.

Примечание — Это преобразование логически завершается до удовлетворения требований последующих разделов.

26.3 Типы "ComponentType" в типе "множество" должны иметь различающиеся теги (см. раздел 30). Тег каждого нового "ComponentType", добавляемого к "AdditionalComponentTypeList", должен быть канонически больше (см. 8.4), чем тег других компонентов в "AdditionalComponentTypeList".

Примечание — Когда "TagDefault" для модуля, в котором появилась данная нотация, есть "AUTOMATIC TAGS", это условие выполняется независимо от фактических типов "ComponentType", в результате применения 24.7.

26.4 Положения 24.2, 24.7—24.13 применяются также и для типов "множество".

26.5 Все типы "множество" имеют тег универсального класса 17.

Примечание — Типы "множество-из" имеют тот же самый тег, что и типы "множество" (см. 27.2).

26.6 Никакая семантика не должна связываться с порядком значений в типе "множество".

26.7 Значение типа "множество" должно определяться нотацией "SetValue":

```
SetValue : : =  "{"  ComponentValueList  "}" | "{"  "}"
```

Продукция "ComponentValueList" определена в 24.16.

26.8 Нотация "{" "}" должна использоваться для "SetValue", только если:

а) все последовательности "ComponentType" в "SetType" помечены как "DEFAULT" или "OPTIONAL", а все значения — опущены, или

б) нотацией типа была "SET {}".

26.9 Должно быть по одному значению "NamedValue" для каждого типа "NamedType" в "SetType", который не отмечен как "DEFAULT" или "OPTIONAL".

Примечание — Эти значения "NamedValue" могут появляться в произвольном порядке.

27 Нотация для типов "множество-из"

27.1 Нотацией для определения типа "множество-из" (см. 3.8.59) из другого типа должна быть "SetOfType":

```
SetOfType : : =
        SET OF Type
```

27.2 Все типы "множество-из" имеют тег универсального класса 17.

Примечание — Типы "множество" имеют тот же самый тег, что и типы "множество-из" (см. 26.5).

27.3 Значение типа "множество-из" должно определяться нотацией "SetOfValue":

```
SetOfValue : : =  "{"  ValueList  "}" | "{"  "}"
```

Список значений "ValueList" определен в 25.3.

Нотация "{" "}" используется, когда SetOfValue есть пустой список.

27.4 Каждое значение "Value" в списке значений "ValueList" должно быть нотацией для значения типа "Type", заданного в "SetOfType".

Примечания

1 Никакое семантическое значение не должно быть связано с порядком этих значений.

2 Не требуется, чтобы правила кодирования сохраняли порядок этих значений.

3 Тип "множество-из" не является математическим множеством значений, например для "SET OF INTEGER" значения "{1}" и "{1,1}" являются различными.

28 Нотация для выборочных типов

28.1 Нотация для определения выборочного типа (см. 3.8.13) из других типов должна быть "ChoiceType":

```
ChoiceType : : = CHOICE "{" AlternativeTypeLists "}"
AlternativeTypeLists : : =
        RootAlternativeTypeList |
        RootAlternativeTypeList "," ExtensionAndException
            ExtensionAdditionAlternatives OptionalExtensionMarker
RootAlternativeTypeList : : = AlternativeTypeList
ExtensionAdditionAlternatives : : =
        "," ExtensionAdditionAlternativesList | empty
ExtensionAdditionAlternativesList : : = ExtensionAdditionAlternative |
        ExtensionAdditionAlternativesList ","
            ExtensionAdditionAlternative
ExtensionAdditionAlternative : : = ExtensionAdditionAlternatives |
        NamedType
ExtensionAdditionAlternatives : : = "[[" AlternativeTypeList "]]"
AlternativeTypeList : : =
        NamedType |
        AlternativeTypeList "," NamedType
```

Примечание — Типы CHOICE : : = { a A } и A являются различными и могут кодироваться различным образом.

28.2 Типы, определенные в продукциях "AlternativeTypeList" в "AlternativeTypeLists", должны иметь различающиеся теги (см. раздел 28). Если действует автоматическое тегирование и "NamedType" в корне расширения не имеют тегов, то "NamedType" в "ExtensionAdditionAlternativesList" не должны тегироваться.

Примечание — Когда "TagDefault" для модуля, в котором появилась данная нотация, есть "AUTOMATIC TAGS", теги автоматически становятся различными в результате применения 24.7.

28.3 Когда продукция "AlternativeTypeLists" встречается в определении модуля, для которого выбрано автоматическое тегирование (см. 12.3), и никакой из экземпляров "NamedType" в ней не содержит тип "Type", являющийся экземпляром "TaggedType", то преобразование автоматического тегирования выбрано для всей "AlternativeTypeList"; в противном случае – нет. Преобразование автоматического тегирования "AlternativeTypeLists", когда оно выбрано, применяется к каждому типу "NamedType" продукции "AlternativeTypeLists" путем замены каждого "Type", исходного для продукции "NamedType", экземпляром замещения "TaggedType", определенного в 24.9.

28.4 Тег каждого нового "NamedType", добавляемого к "ExtensionAdditionAlternativesList", должен быть канонически больше (см. 8.4) тегов других альтернатив в "ExtensionAdditionAlternativesList" и в "ExtensionAdditionAlternativesList" последним должен быть "NamedType".

28.5 Выборочный тип содержит значения, не все из которых имеют один и тот же тег. (Тег зависит от альтернативы, которая дает значение выборочному типу).

28.6 Когда данный тип не имеет маркера расширения и используется там, где настоящий стандарт требует использовать типы с различающимися тегами (см. 24.5, 24.6, 26.3 и 28.2), все возможные теги значений выборочного типа должны рассматриваться с точки зрения этого требования. Это требование иллюстрируется в следующих примерах, где принято, что "TagDefault" не есть "AUTOMATIC TAGS".

Примеры

1
```
A : : = CHOICE
                {b    B,
                 c    NULL}
B : : = CHOICE
                {d    [0] NULL,
                 e    [1] NULL}
```

```
2       A : : = CHOICE
                        {b      B,
                         c      C}
        B : : = CHOICE
                        {d      [0] NULL,
                         e      [1] NULL}
        C : : = CHOICE
                        {f      [2] NULL,
                         g      [3] NULL}
3 (НЕКОРРЕКТНЫЙ)
                A : : = CHOICE
                        {b      B,
                         c      C}
                B : : = CHOICE
                        {d      [0] NULL,
                         e      [1] NULL}
                C : : = CHOICE
                        {f      [0] NULL,
                         g      [1] NULL}
```

В примерах 1 и 2 нотация используется корректно. В примере 3 нотация некорректна без автоматического тегирования, так как теги типов d, f, а также e, g идентичны.

28.7 Идентификаторы "identifier" всех типов "NamedType" в "AlternativeTypeLists" должны отличаться от идентификаторов других типов "NamedType" в этом же списке.

28.8 Значение выборочного типа должно определяться нотацией "ChoiceValue":

```
ChoiceValue : : = identifier ":" Value
```

28.9 "Value" должно быть нотацией для значения того типа в "AlternativeTypeLists", который назван идентификатором "identifier".

29 Нотация для селективных типов

29.1 Нотацией, определяющей селективный тип (см. 3.8.55), должна быть "SelectionType":

```
SelectionType : : = identifier "<" Type
```

где "Type" обозначает выборочный тип, а "identifier" – один из типов "NamedType", появляющихся в "AlternativeTypeLists" определения этого выборочного типа.

29.1.1 Когда "Type" обозначает ограниченный тип, выбор осуществляется из родового типа, игнорируя ограничение.

29.2 Когда "SelectionType" используется в качестве "NamedType", идентификатор "identifier" типа "NamedType" присутствует так же, как и "identifier" типа "SelectionType".

29.3 Когда "SelectionType" используется в качестве "Type", идентификатор "identifier" сохраняется и обозначает тип выбранной альтернативы.

29.4 Нотацией для значения селективного типа должна быть нотация для значения, указанного типом "SelectionType".

30 Нотация для тегированных типов

Тегированный тип (см. 3.8.64) — это новый тип, который является изоморфным старому типу, но имеет другой тег. Тегированный тип используется главным образом там, где настоящим стандартом требуется использование типов с различающимися тегами (см. 24.5, 24.6, 26.3 и 28.2). Использование в модуле "TagDefault" с "AUTOMATIC TAGS" позволяет выполнить эти требования без явной спецификации нотации тегированного типа в этом модуле.

Примечание — Когда протокол определяет, что значения нескольких типов данных могут быть переданы в один и тот же момент времени, различие тегов может быть необходимым для того, чтобы обеспечить получателю возможность корректно декодировать значение.

30.1 Нотацией для тегированного типа должна быть "TaggedType":

```
TaggedType : : =
        Tag Type                    |
        Tag IMPLICIT Type           |
        Tag EXPLICIT Type
Tag : : = "|" Class ClassNumber "|"
ClassNumber : : =
        number  |
        DefinedValue
Class : : =
        UNIVERSAL           |
        APPLICATION         |
        PRIVATE             |
        empty
```

30.2 Ссылка на значение "valuereference" в "DefinedValue" должна быть целочисленного типа, и ей должно быть присвоено неотрицательное значение.

30.3 Новый тип изоморфен старому типу, но имеет тег класса "Class" и номер "ClassNumber", за исключением случая, когда "Class" есть "empty"; в этом случае тег — контекстно зависимого класса с номером "ClassNumber".

30.4 Класс "Class" не может быть универсальным классом "UNIVERSAL", за исключением типов, определенных в настоящем стандарте.

Примечания:

1 Использование тегов универсального класса регулярно согласуется ИСО и МЭК-Т.

2 В подразделе С.2.12 содержится руководство и указания по стилистике использования классов тегов.

30.5 Все применения тегов относятся либо к явному, либо к неявному тегированию. Неявное тегирование указывает (тем правилам кодирования, которые обеспечивают соответствующий выбор), что явная идентификация исходных тегов типа "Type" в "TaggedType" не является необходимой при передаче.

Примечание — Может быть полезным сохранять старый тег, когда он универсального класса и, следовательно, недвусмысленно идентифицирует старый тип без знания определения АСН.1 нового типа. Однако минимальное количество октетов для передачи обычно достигается использованием IMPLICIT. Пример кодирования, использующего IMPLICIT, приведен в ИСО/МЭК 8825-1.

30.6 Конструкция тегирования задает явное тегирование, если выполняется одно из следующих утверждений:

а) используется альтернатива "Tag EXPLICIT Type";

б) используется альтернатива "Tag Type", и значение "TagDefault" для модуля есть либо "EXPLICIT TAGS", любо пусто;

в) используется альтернатива "Tag Type", и значение "TagDefault" для модуля есть либо "IMPLICIT TAGS", любо "AUTOMATIC TAGS", но тип, определяемый "Type", есть выборочный или открытый тип, или тип "DummyReference" (см. ИСО/МЭК 8824-4, 8.3).

30.7 Если "Class" есть "empty", то нет ограничений на использование "Tag", кроме тех, которые подразумеваются требованием различия тегов в 24.5, 24.6, 26.3 и 28.2.

30.8 Альтернатива "IMPLICIT" не должна использоваться, если тип, определяемый "Type", есть выборочный или открытый тип, или тип "DummyReference" (см. ИСО/МЭК 8824-4, 8.3).

30.9 Нотацией для значения "TaggedType" должна быть "TaggedValue":

```
TaggedValue : : =  Value
```

где "Value" есть нотация для значения "Type" в "TaggedType".

Примечание — "Tag" не появляется в этой нотации.

31 Нотация для типа "идентификатор объекта"

31.1 Тип "идентификатор объекта" (см. 3.8.47) должен указываться нотацией "ObjectIdentifierType":

```
ObjectIdentifierType : : =
        OBJECT IDENTIFIER
```

31.2 Этот тип имеет тег универсального класса 6.

31.3 Нотацией для значения идентификатора объекта должна быть "ObjectIdentifierValue":

```
ObjectIdentifierValue : : =
        "{" ObjIdComponentList "}"  |
        "{" DefinedValue ObjIdComponentList "}"
ObjIdComponentList . : =
        ObjIdComponent                  |
        ObjIdComponent ObjIdComponentList
ObjIdComponent : : = NameForm           |
                     NumberForm         |
                     NameAndNumberForm
NameForm : : =  identifier
NumberForm : : = number  | DefinedValue
NameAndNumberForm : : =
          identifier "(" NumberForm ")"
```

31.4 Ссылка на значение "valuereference" в "DefinedValue" числовой формы "NumberForm" должна быть целочисленного типа и ей должно быть присвоено неотрицательное значение.

31.5 Ссылка на значение "valuereference" в "DefinedValue" как на значение идентификатора объекта "ObjectIdentifierValue" должна быть типа "идентификатор объекта".

31.6 Именная форма "NameForm" должна использоваться только для тех компонентов идентификатора объекта, численное значение и идентификатор которых определены в приложениях А — С ИСО/МЭК 9834-1, и должна быть одним из идентификаторов, определенных в этих приложениях. Когда ИСО/МЭК 9834-1 определяет синонимы идентификаторов, любой синоним может использоваться с той же самой семантикой. Когда одно и то же имя является идентификатором, определенным в ИСО/МЭК 9834-1, и значением ссылки ACH.1 в модуле, содержащем "NameForm", имя в значении идентификатора объекта должно трактоваться как идентификатор ИСО/МЭК 9834-1.

31.7 Альтернатива "number" в числовой форме "NumberForm" должна быть числовым значением, присвоенным компоненту идентификатора объекта.

31.8 Идентификатор "identifier" в "NameAndNumberForm" должен быть задан, когда числовое значение присвоено компоненту идентификатора объекта.

Примечание — Уполномоченные, распределяющие численные значения компонентам идентификаторов объектов, идентифицированы в ИСО/МЭК 9834-1.

31.9 Семантика значения идентификатора объекта определяется в ИСО/МЭК 9834-1.

31.10 Значащей частью компонента идентификатора объекта является "NameForm" или "NumberForm", к которой он сводится и которая предоставляет числовое значение для компонента идентификатора объекта. За исключением дуг, определенных в приложениях А – С ИСО/МЭК 9834-1, числовое значение компонента идентификатора объекта всегда присутствует в экземпляре нотации значения идентификатора объекта.

31.11 Когда значение идентификатора объекта "ObjectIdentifierValue" содержит альтернативу "DefinedValue", список компонентов идентификатора объекта, на который она ссылается, предшествует компонентам, явно присутствующим в значении.

Примечание — ИСО/МЭК 9834-1 рекомендует, чтобы при присваивании значения идентификатора объекта также присваивалось и значение описателя объекта.

Примеры

С идентификаторами, присвоенными в ИСО/МЭК 9834-1, значения

{iso standard 8571 pci (1)}

и

{1 0 8571 1}

идентифицируют объект "pci", определенный в ИСО 8571.

С дополнительным определением

ftam OBJECT IDENTIFIER : : = {iso standard 8571}

этим значениям эквивалентно следующее:

{ftam pci (1)}

32 Нотация для типа "встроенное-здп"

32.1 Тип "встроенное-здп" (см. 3.8.21) должен указываться нотацией "EmbeddedPDVType":

EmbeddedPDVType : : = EMBEDDED PDV

32.2 Этот тип имеет тег универсального класса 11.

Примечание — Когда используется согласование уровня представления, те же самые функциональные возможности, что и EXTERNAL, обеспечиваются EMBEDDED PDV (вместе с дополнительными функциональными возможностями), но биты в строке будут другими. В этом случае рекомендуется, чтобы в последующих изменениях версии прикладного протокола была сделана замена EXTERNAL CHOICE {external EXTERNAL, embedded-pdv EMBEDDED PDV}. Дополнительные замены использования EXTERNAL там, где не используется согласование уровня представления, предлагаются в ГОСТ Р ИСО/МЭК 8824-2, приложение С.

32.3 Тип состоит из значений, представляющих:

а) кодирование единственного значения данных, которое может быть, а может и не быть, значением типа ACH.1, и

б) идентификацию (отдельно или вместе):

1) класса значений, содержащего это значение данных (абстрактного синтаксиса), и

2) кодирования, использованного для отличия этого значения данных от других того же класса (синтаксиса передачи).

Примечания

1 Значение данных может быть значением типа ACH.1 или, например, кодированием неподвижного или движущегося изображения. Идентифицируются либо один или два идентификатора объектов, либо ссылки на контекст представления ВОС для идентификации абстрактного синтаксиса и синтаксиса передачи.

2 Идентификация абстрактного синтаксиса и/или кодирования может также определяться проектировщиком приложения в качестве фиксированного значения, и в этом случае она может не кодироваться в конкретном сеансе взаимосвязи.

32.4 Тип "встроенное-здп" имеет ассоциированный тип. Этот тип используется для обеспечения нотации значений и подтипов типа "встроенное-здп".

32.5 Ассоциированный тип для определения значения и подтипа, принимая окружение автоматического тегирования, имеет вид (с нормативными комментариями):

```
SEQUENCE {
    identification              CHOICE {
        syntaxes                        SEQUENCE {
            abstract                    OBJECT IDENTIFIER,
            transfer                    OBJECT IDENTIFIER }
            - - Идентификаторы объектов абстрактного синтаксиса и синтаксиса передачи - -,
        syntax                  OBJECT IDENTIFIER
            - - Единственный идентификатор объекта для идентификации класса или
            - - кодирования - -,
        presentation-context-id INTEGER
            - - (Применяется только в среде ВОС)
            - - Согласованный контекст уровня представления
            - - идентифицирует класс значения и его кодирование - - ,
        context-negotiation     SEQUENCE {
            presentation-context-id     INTEGER,
            transfer-syntax             OBJECT IDENTIFIER }
            - - (Применяется только в среде ВОС)
            - - Осуществляется процесс согласования контекста для идентификации класса
            - - значения и его кодирования - -,
        transfer-syntax         OBJECT IDENTIFIER
            - - Класс значения (например спецификация того, что это значение типа ACH.1)
            - - зафиксирован разработчиком приложения (и, следовательно, известен как
            - - отправителю, так и получателю). Этот случай предназначен главным образом для
```

```
        - - обеспечения выборочного шифрования поля (или другого преобразования
        - - кодирования) типа ACH.1 - -,
    fixed                   NULL
        - - Значение данных есть значение фиксированного типа ACH.1 (и, следовательно,
        - - известно как отправителю, так и получателю) - - },
data-value-descriptor       ObjectDescriptor OPTIONAL
        - - Этим обеспечивается человекочитаемая идентификация класса значения - -,
data-value                  OCTET STRING}
(WITH COMPONENTS {
    ... ,
    data-value-descriptor ABSENT})
```

Примечание — Тип "встроенное-зпд" не допускает включения значения "data-value-descriptor". Однако определение ассоциированного типа отражает базовую общность, которая существует между типом "встроенное-здп", внешним типом и неограниченным типом символьных строк.

32.6 Для альтернативы "presentation-context-id" целочисленное значение должно быть идентификатором контекста представления в множестве определенных контекстов. Эта альтернатива не должна использоваться ни в запросе P-CONNECT, ни в запросе P-ALTER-CONNECT для контекста представления, который предлагается для добавления или удаления этими примитивами запросов.

Примечание — Даже если предлагается единственный синтаксис передачи для контекста представления в списке определений контекстов представления, альтернатива "presentation-context-id" не может использоваться для контекста представления.

32.7 Альтернатива "context-negotiation" должна использоваться в запросах P-CONNECT или в P-ALTER-CONNECT, а целочисленное значение должно быть идентификатором контекста представления, предложенного для добавления к множеству определенных контекстов. Идентификатор объекта "transfer-syntax" должен идентифицировать предложенный синтаксис передачи для того контекста представления, который используется для кодирования значения.

32.8 Нотация для значения типа "встроенное-здп" должна быть нотацией значения для ассоциированного типа, определенного в 32.5, где значение "data-value" OCTET STRING представляет собой кодирование, использующее специфицированный в "identification" синтаксис передачи.

EmbeddedPdvValue : : = SequenceValue - - значение ассоциированного типа, определенного в 32.5.

32.9 Пример 1 — Когда проектировщик приложения хочет, чтобы кодирование не зависело от любого окружения представления (и, следовательно, могло передаваться, храниться и вызываться без модификаций), необходимо запретить использование альтернатив "presentation-context-id" и "context-negotiation". Это может быть сделано следующим образом:

```
EMBEDDED PDV (WITH COMPONENTS {
        ... ,
        identification (WITH COMPONENTS {
            ... ,
            presentation-context-id      ABSENT,
            context-negotiation          ABSENT })})
```

32.10 Пример 2 — Если обязательно должна использоваться единственная альтернатива, например "syntaxes", то это может быть сделано следующим образом:

```
EMBEDDED PDV (WITH COMPONENTS {
        ... ,
        identification (WITH COMPONENTS {
          syntaxes PRESENT })})
```

33 Нотация для внешнего типа

33.1 Внешний тип (см. 3.8.37) должен указываться нотацией "ExternalType":

ExternalType : : = EXTERNAL

33.2 Этот тип имеет тег универсального класса 8.

33.3 Тип состоит из значений, представляющих:

а) кодирование единственного значения данных, которое может быть, а может и не быть значением типа ACH.1, и

б) идентификацию:

1) класса значений, содержащего это значение данных (абстрактного синтаксиса), и

2) кодирования, использованного для отличия этого значения данных от других того же класса (синтаксиса передачи), и

в) (факультативно) описатель объекта, который обеспечивает человекочитаемое описание класса значения данных. Факультативный описатель объекта может присутствовать, если только он явно допускается комментарием, связанным с использованием нотации "ExternalType".

Примечание — Примечание 1 к 32.3 применяется так же и к внешнему типу.

33.4 Внешний тип имеет ассоциированный тип. Он используется для задания точного определения абстрактных значений внешнего типа и обеспечения нотаций для его значения и подтипа.

Примечание — Правила кодирования могут определять различные типы, которые используются для производного кодирования, или могут специфицировать кодирования без ссылки на какой-либо ассоциированный тип. В частности, кодирование BER использует эквивалент типа "последовательность", идентичный тому, который присутствовал в определении внешнего типа в ГОСТ Р ИСО/МЭК 8824, и кодирование внешних значений с помощью BER не изменяется.

33.5 Ассоциированный тип для определения значения и подтипа, принимая окружение автоматического тегирования, имеет вид (с нормативными комментариями):

```
SEQUENCE {
    identification              CHOICE {
        syntaxes                        SEQUENCE {
            abstract                    OBJECT IDENTIFIER,
            transfer                    OBJECT IDENTIFIER }
            - - Идентификаторы объектов абстрактного синтаксиса и синтаксиса передачи - -,
        syntax                  OBJECT IDENTIFIER
            - - Единственный идентификатор объекта для идентификации класса или
            - - кодирования - -,
        presentation-context-id INTEGER
            - - (Применяется только в среде ВОС)
            - - Согласованный контекст уровня представления
            - - идентифицирует класс значения и его кодирование - - ,
        context-negotiation     SEQUENCE {
            presentation-context-id     INTEGER,
            transfer-syntax             OBJECT IDENTIFIER }
            - - (Применяется только в среде ВОС)
            - - Осуществляется процесс согласования контекста для идентификации класса
            - - значения и его кодирования - -,
        transfer-syntax         OBJECT IDENTIFIER
            - - Класс значения (например спецификация того, что это значение типа ACH.1)
            - - зафиксирован разработчиком приложения (и, следовательно, известен как
            - - отправителю, так и получателю). Этот случай предназначен главным образом для
            - - обеспечения выборочного шифрования поля (или другого преобразования
            - - кодирования) типа ACH.1 - -,
        fixed                   NULL
            - - Значение данных есть значение фиксированного типа ACH.1 (и, следовательно,
            - - известно как отправителю, так и получателю) - - },
    data-value-descriptor       ObjectDescriptor OPTIONAL
            - - Этим обеспечивается человекочитаемая идентификация класса значения - -,
    data-value                  OCTET STRING}
```

```
(WITH COMPONENTS {
        ... ,
        identification (WITH COMPONENTS {
            ... ,
            syntaxes                ABSENT,
            transfer-syntax         ABSENT,
            fixed                   ABSENT })})
```

Примечание — Внешний тип не допускает включения альтернатив "syntaxes", "transfer-syntax" и "fixed" для "identification". Эти альтернативы не могут быть допустимы для внешнего типа потому, что необходимо поддерживать обратную совместимость с ГОСТ Р ИСО/МЭК 8824. Проектировщики приложений, которым требуются эти альтернативы, должны использовать тип "встроенное-здп". Определение ассоциированного типа отражает базовую общность, которая существует между типом "встроенное-зпд", внешним типом и неограниченным типом символьных строк.

33.6 Положения 32.6 и 32.7 применяются также и к внешнему типу.

33.7 Нотация для значения внешнего типа должна быть нотацией значения для ассоциированного типа, определенного в 33.5, где значение "data-value" OCTET STRING представляет собой кодирование, использующее специфицированный в "identification" синтаксис передачи.

ExternalValue : : = SequenceValue - - значение ассоциированного типа, определенного в 33.5.

Примечание — По историческим причинам правила кодирования могут передавать в EXTERNAL встроенные значения, кодирования которых не являются кратными 8 битам. Такие значения не могут быть представлены в нотации значения, использующей ассоциированный тип.

34 Типы символьных строк

Эти типы состоят из строк символов из некоторого заданного символьного репертуара. Обычно определяют символьный репертуар и его кодирование с использованием ячеек одной или нескольких таблиц; каждая ячейка соответствует символу в репертуаре. Обычно каждой ячейке присвоены графический символ и имя символа, хотя в некоторых репертуарах ячейки оставлены пустыми или имеют имена, но не имеют представления (например ячейки с именами, но без представления, включают управляющие символы, как EOF в ИСО 646, и символы интервала, как THIN-SPACE и EN-SPACE в ИСО/МЭК 10646-1).

Термин абстрактный символ обозначает всю информацию, связанную с ячейкой в таблице символьного репертуара. Информация, связанная с ячейкой, обозначает отдельный абстрактный символ в репертуаре, даже если этой информацией является null (ни графический символ, ни имя не присвоены ячейке).

Нотация значений ACH.1 для типов символьных строк имеет три варианта (которые можно комбинировать), формально определенные ниже:

а) печатное представление символов в строке, использующее присвоенные графические символы, возможно, включая символы интервала; это нотация "cstring":

Примечания

1 Такое представление может оказаться двусмысленным, когда один и тот же графический символ используется для нескольких символов в репертуаре.

2 Такое представление может оказаться двусмысленным, когда используются символы интервала или спецификация печатается с использованием пропорционального шрифта;

б) перечень символов в значении символьной строки путем задания серий ссылок на значения ACH.1, которые были присвоены символам; множество таких ссылок на значения определено в модуле ASN1-CHARACTER-MODULE в разделе 37 для репертуаров символов ИСО/МЭК 10646-1 и IA5String; это представление не доступно для других символьных репертуаров, если только пользователь не определит такие ссылки на значения, используя нотацию, описанную в а) или в);

в) перечень символов в значении символьной строки путем идентификации каждого абстрактного символа позицией его ячейки в таблице репертуара символов; эта форма доступна только для IA5String, UniversalString, UTF8String и BMPString.

35 Нотация для типов символьных строк

35.1 Нотацией для указания типа символьной строки (см. 3.8.11) должна быть

```
CharacterStringType : : =  RestrictedCharacterStringType  |
                           UnrestrictedCharacterStringType
```

"RestrictedCharacterStringType" является нотацией для ограниченного типа символьной строки и определена в разделе 36; "UnrestrictedCharacterStringType" является нотацией для неограниченного типа символьной строки и определена в 39.1.

35.2 Тег каждого ограниченного типа символьной строки определен в 36.1. Тег неограниченного типа символьной строки определен в 39.2.

35.3 Нотацией для значения символьной строки должна быть

```
CharacterStringValue : : = RestrictedCharacterStringValue  |
                           UnrestrictedCharacterStringValue
```

Нотация "RestrictedCharacterStringValue" определена в разделе 36.7. Нотация "UnrestrictedCharacterStringValue" определена в 39.6.

36 Определение ограниченных типов символьных строк

В данном разделе определяются типы, значения которых ограничены последовательностями из нуля, одного или нескольких символов из некоторой заданной совокупности символов. Нотацией для указания ограниченного типа символьной строки должна быть "RestrictedCharacterStringType":

```
RestrictedCharacterStringType : : = BMPString       |
                                    GeneralString   |
                                    GraphicString   |
                                    IA5String       |
                                    ISO646String    |
                                    NumericString   |
                                    PrintableString |
                                    TeletexString   |
                                    T61String       |
                                    UniversalString |
                                    UTF8String      |
                                    VideotexString  |
                                    VisibleString
```

Каждая альтернатива "RestrictedCharacterStringType" определяется заданием:

а) тега, присвоенного типу, и

б) имени (например NumericString), которым тип указывается, и

в) символами в совокупности символов, используемой в определении типа, путем указания таблицы, перечисляющей графические символы, или через ссылку на регистрационный номер в Международном регистре ИСО наборов кодированных символов, или через ссылку на ИСО/МЭК 10646-1.

36.1 В таблице 3 приведены имена, по которым ссылаются на ограниченные типы символьных строк, номера тегов универсального класса, присвоенные типам, определяющие регистрационные номера, таблицы или номера разделов настоящего стандарта и, при необходимости, идентифицированы примечания, относящиеся к строке таблицы. Когда в нотации определены синонимы, они приводятся в скобках.

Примечание — Теги, присвоенные типам символьных строк, недвусмысленно идентифицируют тип. Однако если АСН.1 используется для определения новых типов из приведенных здесь (в частности, используя IMPLICIT), то будет невозможно распознать эти типы без знания определения АСН.1.

Таблица 3 — Перечень зарегистрированных типов символьных строк

Имя для ссылки на тип	Номер тега универсального класса	Определяющий регистрационный номер*), номер таблицы или раздела стандарта	Примечание
UTF8String	12	См. 36.13	—
NumericString	18	Таблица 4	1
PrintableString	19	Таблица 5	1
TeletexString (T61Strng)	20	6, 87, 102, 103, 106, 107, 126, 144, 150, 153, 156, 164, 165, 168 + SPACE + DELETE	2
VideotexString	21	1, 13, 72, 73, 87, 89, 102, 108, 126, 128, 129, 144, 150, 153, 164, 165, 168 + SPACE + DELETE	3
IA5String	22	1, 6 + SPACE + DELETE	—
GraphicString	25	Все графические наборы + SPACE	—
VisibleString (IS0646Strin)	26	6 + SPACE	4
GeneralStrng	27	Все графические и символьные наборы + SPACE + DELETE	—
UniversalString	28	См. 36.6	—
BMPString	30	См. 36.12	—

*) Определяющие регистрационные номера приведены в Международном регистре ИСО наборов кодированных символов, которые должны использоваться с Escape-последовательностями.

Примечания

1 Тип-стиль, размер, цвет, интенсивность и прочие характеристики отображения не существенны.

2 Записи, соответствующие этим регистрационным номерам, ссылаются на рекомендацию МСЭ-Т Т.61, содержащую правила их использования. Регистрационные записи 6 и 156 могут использоваться вместо записей 102 и 103.

3 Записи, соответствующие этим регистрационным номерам, обеспечивают функциональные возможности Рекомендаций МСЭ-Т Т.100 и Т.101.

4 Указание регистрационного номера 6 Международного регистра ИСО наборов кодированных символов, которые должны использоваться с Escape-последовательностями, является неявным указанием ИСО 646. В этом состоит отличие от ГОСТ Р ИСО/МЭК 8824, где приводился регистрационный номер 2 (неявное указание на ИСО 646). В приложениях, где желательно применять регистрационный номер 2, следует использовать другие способы указания [например через неограниченные символьные строки (см. раздел 39)] для обращения к старому определению VisibleString или к ГОСТ Р ИСО/МЭК 8824.

36.2 В таблице 4 приведены символы, которые могут появляться в типе NumericString, и их абстрактный синтаксис.

36.3 Следующие значения идентификатора и описателя объекта назначены для идентификации и описания абстрактного синтаксиса символов NumericString:

{ joint-iso-itu-t asn1 (1) specification (0) characterStrings (1) numericString (0) }

и

"NumericString character abstract syntax"

Таблица 4 — Тип NumericStrng

Имя	Графическое представление
Цифры	0, 1, . . . 9
Пробел	(пробел)

Примечания

1 Это значение идентификатора объекта может использоваться в значениях CHARACTER STRING и в других случаях, когда необходимо передать идентификацию типа символьной строки отдельно от значения.

2 Значение абстрактного синтаксиса символов NumericString может быть закодировано:

а) по одному из правил ИСО/МЭК 10646-1 для кодирования абстрактных символов. В этом случае синтаксис передачи символов идентифицируется идентификатором объекта, связанным с этими правилами в ИСО/МЭК 10646-1, приложение М;

б) по правилам кодирования АСН.1 для встроенного типа NumericString. В этом случае синтаксис передачи символов идентифицируется значением идентификатора объекта { joint-iso-itu-t asn1 (1) basic-encoding (1) }.

36.4 В таблице 5 приведены символы, которые могут появляться в типе PrintableString, и их абстрактный синтаксис.

Таблица 5 — Тип PrintableString

Имя	Графическое представление	Имя	Графическое представление
Прописные буквы	A, B, ... Z	Запятая	,
Строчные буквы	a, b, ... z	Дефис	-
Цифры	0, 1, ... 9	Точка	.
Пробел	(пробел)	Наклонная черта	/
Апостроф	'	Двоеточие	:
Левая скобка	(	Знак равенства	=
Правая скобка	)	Знак вопроса	?
Знак плюс	+		

36.5 Следующие значения идентификатора и описателя объекта назначены для идентификации и описания абстрактного синтаксиса символов PrintableString:

{ joint-iso-itu-t asn1 (1) specification (0) characterStrings (1) printableString (1) }

и

"PrintableString character abstract syntax"

Примечания

1 Это значение идентификатора объекта может использоваться в значениях CHARACTER STRING и в других случаях, когда необходимо передать идентификацию типа символьной строки отдельно от значения.

2 Значение абстрактного синтаксиса символов PrintableString может быть закодировано:

а) по одному из правил ИСО/МЭК 10646-1 для кодирования абстрактных символов. В этом случае синтаксис передачи символов идентифицируется идентификатором объекта, связанным с этими правилами в ИСО/МЭК 10646-1, приложение М;

б) по правилам кодирования АСН.1 для встроенного типа PrintableString. В этом случае синтаксис передачи символов идентифицируется значением идентификатора объекта {joint-iso-itu-t asn1(1) basic-encoding (1)}.

36.6 Символами, которые могут появляться в типе UniversalString, являются любые символы, допускаемые ИСО/МЭК 10646-1, а использование этого типа влечет за собой требования соответствия, специфицированные в ИСО/МЭК 10646-1, особенно относительно ограничений на использование зон ИСО/МЭК 10646-1.

Примечания

1 Использование этого типа без ограничений не рекомендуется, так как соответствие будет, в общем случае, не практичным.

2 В разделе 37 определен модуль АСН.1, содержащий ряд подтипов этого типа для совокупности графических символов для подмножеств, определенной в приложении А к ИСО/МЭК 10646-1.

36.7 Нотацией значений для ограниченных типов символьных строк должна быть "cstring" (см. 11.11), "CharacterStringList", "Quadruple" или "Tuple". Нотация "Quadruple" допускается только для символьных строк длиной в один символ и может использоваться только в нотации значения для типов UniversalString, UTF8String или BMPString. Нотация "Tuple" допускается только для символьных строк длиной в один символ и может использоваться только в нотации значения для типа IA5String.

```
RestrictedCharacterStringValue : : =  cstring | CharacterStringList | Quadruple | Tuple
CharacterStringList : : = "{" CharSyms "}"
CharSyms : : = CharsDefn | CharSyms "," CharsDefn
CharsDefn : : = cstring | DefinedValue
Quadruple : : = "{" Group "," Plane "," Row "," Cell "}"
Group . : = number
Plane  . : = number
Row    : : = number
Cell   : : = number
Tuple  : : = "{" TableColumn "," TableRow "}"
TableColumn : : = number
TableRow . . = number
```

Примечания

1 Нотация "cstring" может использоваться только для среды, допускающей отображение графических символов, представленных в значении. Обратно, если среда не имеет таких возможностей, то единственным способом спецификации значения символьной строки, использующей такие графические символы, является нотация "CharacterStringList", только если тип есть UniversalString, UTF8String, BMPString или IA5String; при этом для "CharsDefn" используется альтернатива "DefinedValue" (см. 37.1.2).

2 В разделе 37 определяется ряд ссылок "valuereference", которые обозначают единичные символы (строки длиной 1) типов BMPString (и, следовательно, UniversalString и UTF8String) и IA5String.

Пример — Допустим, что мы хотим специфицировать значение "abcΣdef" для UniversalString, когда символ "Σ" не может быть представлен в доступной среде; это значение может быть выражено как: IMPORTS BasicLatin, greekCapitalLetterSigma FROM ASN1-CHARACTER-MODULE

{joint-iso-itu-t asn1 (1) specification (0) modules (0) iso10646 (0)}; MyAlphabet : : = UniversalString (FROM (BasicLatin | greekCapitalLetterSigma)) mystring MyAlphabet : : = {"abc", greekCapitalLetterSigma, "def"}

3 При спецификации значения типа UniversalString, UTF8String или BMPString, нотация "cstring" не должна использоваться, если двусмысленность, возникающая из-за различия графических символов с одинаковым представлением, не может быть разрешена.

Пример — Следующая нотация "cstring" не должна использоваться, так как графические символы 'H', 'O', 'P' и 'E' встречаются в алфавитах BASIC LATIN, CYRILLIC и BASIC GREEK и, следовательно, двусмысленны.

IMPORTS BasicLatin, Cyrillic, BasicGreek FROM ASN1-CHARACTER-MODULE

{joint-iso-itu-t asn1 (1) specification (0) modules (0) iso10646 (0)}; MyAlphabet : : = UniversalString (FROM (BasicLatin | Cyrillic | BasicGreek)) mystring MyAlphabet : : = "HOPE"

Альтернативным недвусмысленным определением "mystring" могло бы быть следующее:

mystring MyAlphabet (BasicLatin) : : = "HOPE"

Формально "mystring" является ссылкой на значение подмножества "MyAlphabet", но она, по правилам отображения значений, приведенным в приложении F, может быть использована, когда необходима ссылка на значение в пределах "MyAlphabet".

36.8 Альтернатива "DefinedValue" в "CharsDefn" должна ссылаться на значение этого типа.

36.9 Число "number" в продукциях "Plane", "Row" и "Cell" должно быть меньше 256, а в продукции "Group" – меньше 128.

36.10 Продукция "Group" специфицирует группу в кодовом пространстве UCS, "Plane" — плоскость в группе, "Row" — строку в плоскости, а "Cell" — ячейку в строке. Абстрактный символ, идентифицированный этой нотацией, является абстрактным символом, специфицированным значениями "Group", "Plane", "Row" и "Cell". В любом случае множество допустимых значений может быть ограничено введением подтипа.

Примечание — Проектировщики приложений должны тщательно рассматривать соответствие, когда используют такие типы символьных строк с открытым завершением, как GeneralString, GraphicString и UniversalString без ограничений. Тщательное рассмотрение соответствия необходимо и для ограниченных, но длинных типов строк, таких как TeletexString.

36.11 Число "number" в продукции "TableColumn" должно быть от нуля до семи, а в продукции "TableRow" — от нуля до пятнадцати. "TableColumn" задает столбец, а "TableRow" – строку в кодовой таблице в соответствии с рисунком 1 ИСО/МЭК 2022. Эта нотация используется только для IA5String, когда кодовая таблица содержит регистрационную запись 1 в столбцах 0, 1 и регистрационную запись 6 в столбцах 2—7 (см. Международный регистр ИСО наборов кодированных символов, которые должны использоваться с Escape-последовательностями).

36.12 BMPString является подтипом UniversalString, который имеет свой собственный уникальный тег и моделирует основную многоязычную плоскость (первые 64К — 2 ячеек) ИСО/МЭК 10646-1. Он имеет ассоциированный тип, определенный как:

UniversalString (Bmp)

где Bmp определяется в модуле ACH.1 ASN1-CHARACTER-MODULE (см. раздел 37) как подтип UniversalString, соответствующий совокупности имен "BMP", определенной в ИСО/МЭК 10646-1, приложение А.

Примечания

1 Так как BMPString является встроенным типом, то он не определяется в ASN1-CHARACTER-MODULE.

2 Целью определения BMPString как встроенного типа является предоставление возможности по правилам кодирования (таким как BER), которые не учитывают ограничения, использовать 16-битовое, а не 32-битовое кодирование.

3 В нотациях значений BMPString допустимы значения UniversalString и UTF8String.

36.13 На абстрактном уровне UTF8String является синонимом UniversalString и может применяться всякий раз, когда используется UniversalString (подчиняясь правилам, требующим различия тегов), но имеет другой тег и тип.

Примечание — Его кодирование отличается от кодирования UniversalString и в большинстве случаев будет менее длинным.

37 Наименование символов и совокупностей, определенных в ИСО/МЭК 10646-1

В настоящем разделе специфицирован встроенный модуль ACH.1, который содержит определения ссылочных имен значений для всех символов ИСО/МЭК 10646-1, и каждое имя указывает значение UniversalString длиной 1. Этот модуль также содержит определения ссылочных имен типов для всех совокупностей символов ИСО/МЭК 10646-1, и каждое имя указывает подмножество UniversalString.

Примечание — Эти значения доступны для использования в нотациях значений типа UniversalString и типов, полученных из него. Все ссылки на типы и значения, определенные в модуле, специфицированном в 37.1, являются экспортируемыми и должны быть импортированы любым использующим их модулем.

37.1 Спецификация модуля ACH.1 "ASN1-CHARACTER-MODULE"

Данный модуль не приводится здесь полностью. Вместо этого специфицирован способ, которым он определяется.

37.1.1 Модуль начинается следующим образом:

```
ASN1-CHARACTER-MODULE {joint-iso-itu-t asn1 (1) specification (0) modules (0) iso10646 (0)}
DEFINITIONS : : = BEGIN
- - Все ссылки на значения и типы, определенные в данном модуле, являются
- - экспортируемыми и могут быть импортированы любым модулем.
- - Управляющие символы ИСО 646:
nul  IA5String : : = {0, 0}
soh  IA5String : : = {0, 1}
stx  IA5String : : = {0, 2}
etx  IA5String : : = {0, 3}
eot  IA5String : : = {0, 4}
enq  IA5String : : = {0, 5}
ack  IA5String : : = {0, 6}
bel  IA5String : : = {0, 7}
bs   IA5String : : = {0, 8}
ht   IA5String : : = {0, 9}
lf   IA5String : : = {0, 10}
vt   IA5String : : = {0, 11}
ff   IA5String : : = {0, 12}
cr   IA5String : : = {0, 13}
so   IA5String : : = {0, 14}
si   IA5String : : = {0, 15}
dle  IA5String : : = {1, 0}
```

dc1 IA5String : : = {1, 1}
dc2 IA5String : : = {1, 2}
dc3 IA5String : : = {1, 3}
dc4 IA5String : : = {1, 4}
nak IA5String : : = {1, 5}
syn IA5String : : = {1, 6}
etb IA5String : : = {1, 7}
can IA5String : : = {1, 8}
em IA5String : : = {1, 9}
sub IA5String : : = {1, 10}
esc IA5String : : = {1, 11}
is4 IA5String : : = {1, 12}
is3 IA5String : : = {1, 13}
is2 IA5String : : = {1, 14}
is1 IA5String : : = {1, 15}
del IA5String : : = {7, 15}

37.1.2 Для каждой записи в каждом списке имен для графических символов (глифов), показанных в разделах 24 и 25 ИСО/МЭК 10646-1, модуль содержит утверждение вида:

<namedcharacter>BMPString : : = <tablecell>
- - представляет символ <iso10646name>, см. ИСО/МЭК 10646-1,

где: а) <iso10646name> – имя символа, полученное из перечисленных в ИСО/МЭК 10646-1;

б) <namedcharacter> — строка, полученная применением к <iso10646name> процедур, установленных в 37.2;

в) <tablecell> – глиф в ячейке таблицы в ИСО/МЭК 10646-1, соответствующей записи списка.

Пример

latinCapitalLetterA BMPString : : = {0, 0, 0, 65}
- - представляет символ LATIN CAPITAL LETTER A, см. ИСО/МЭК 10646-1
greekCapitalLetterSigma BMPString . : = {0, 0, 3, 145}
- - представляет символ GREEK CAPITAL LETTER SIGMA, см. ИСО/МЭК 10646-1.

37.1.3 Для каждого имени совокупности графических символов, определенной в ИСО/МЭК 10646-1, приложение А, в модуль включается утверждение вида:

<namedcollectionstring> : : = BMPString
(FROM (<alternativelist>))
- - представляет совокупность символов <collectionstring>,
- - см. ИСО/МЭК 10646-1,

где а) <collectionstring> -- имя совокупности, присвоенное в ИСО/МЭК 10646-1;

б) <namedcollectionstring> — образовано применением к <collectionstring> процедуры 37.3;

в) <alternativelist> — образуется с использованием <namedcharacter>, как описано в 37.2, для каждого символа, определенного в ИСО/МЭК 10646-1.

Результирующая ссылка на тип, <namedcollectionstring>, образует ограниченное подмножество (см. руководство в приложении D).

Примечание – Ограниченное подмножество является списком символов в заданном подмножестве. Противоположность ему – выбранное подмножество, которое является совокупностью символов, перечисленных в ИСО/МЭК 10646-1, приложение А, плюс совокупность BASIC LATIN.

Пример (частичный):

```
space BMPString : : = {0, 0, 0, 32}
exclamationMark BMPString : : = {0, 0, 0, 33}
quotationMark BMPString . : = {0, 0, 0, 34}
...          - - и так далее
tilde BMPString : : = {0, 0, 0, 126}
BasicLatin : : = BMPString
    (FROM (space
    |exclamationMark
```

```
        |quotationMark
        |...        - - и так далее
        |tilde)
        )
    - - представляет совокупность символов BASIC LATIN, см. ИСО/МЭК 10646-1.
    - - Многоточия в этом примере используются для краткости и означают
    - - "и так далее";
    - - их нельзя использовать в реальном модуле ACH.1.
```

37.1.4 В ИСО/МЭК 10646-1 определены три уровня реализации. По умолчанию все типы, определенные в модуле ASN1-CHARACTER-MODULE, за исключением "Level1" и "Level2", соответствуют реализации уровня 3, так как эти типы не имеют ограничений на использование комбинированных символов. "Level1" указывает, что требуется реализация уровня 1, "Level2" – уровня 2, а "Level3" – уровня 3. Таким образом, в ASN1-CHARACTER-MODULE определено следующее:

```
Level1 : : = BMPString (FROM (ALL EXCEPT CombiningCharacters))
Level2 : : = BMPString (FROM (ALL EXCEPT CombiningCharactersB-2))
Level3 : : = BMPString
```

Примечания

1 "CombiningCharacters" и "CombiningCharactersB-2" являются ссылками <namedcollectionstring>, удовлетворяющими совокупностям "COMBINING CHARACTERS" и "COMBINING CHARACTERS B-2", соответственно, определенным в ИСО/МЭК 10646-1, приложение А.

2 "Level1" и "Level2" используются либо следом за "IntersectionMark" (см. раздел 46), либо как единственное ограничение в "ConstraintSpec". Примеры см. в С.2.7.1.

3 Дополнительную информацию см. в D.2.5.

37.1.5 Модуль завершается утверждением:

```
END
```

37.1.6 Определяемым пользователем эквивалентом примера 37.1.3 является:

```
BasicLatin : : = BMPString (FROM (space . . tilde))
- - представляет совокупность символов BASIC LATIN, см. ИСО/МЭК 10646-1.
```

37.2 <namedcharacter> является строкой, полученной из <iso10646name> (см. 37.1.2) применением следующего алгоритма:

а) каждая прописная буква <iso10646name> преобразуется в соответствующую строчную букву, если только прописной букве не предшествует символ SPACE; в этом случае прописная буква остается неизменной;

б) каждая цифра и символ HYPHEN-MINUS остаются неизменными;

в) каждый символ SPACE удаляется.

Примечание — Приведенный алгоритм, вместе с руководством по наименованию символов в приложении К ИСО/МЭК 10646-1, всегда приведет к недвусмысленной нотации значения для любого имени символа, приведенного в ИСО/МЭК 10646-1.

Пример — Символ ИСО/МЭК 10646-1 в строке 0, ячейке 60, который назван "LESS-THAN SIGN" и имеет графическое представление "<", может быть указан с использованием "DefinedValue"

```
less-thanSign
```

37.3 <namedcollectionstring> является строкой, полученной из <collectionstring> применением следующего алгоритма:

а) каждая прописная буква в имени совокупности ИСО/МЭК 10646-1 преобразуется в соответствующую строчную букву, если только прописной букве не предшествует символ SPACE или она не является первой буквой в имени; в этом случае прописная буква остается неизменной;

б) каждая цифра и символ HYPHEN-MINUS остаются неизменными;

в) каждый символ SPACE удаляется.

Примеры

1 Совокупность, идентифицированная в приложении А ИСО/МЭК 10646-1 как

```
BASIC LATIN
```

имеет ссылку на тип ACH.1

```
BasicLatin
```

2 Тип символьной строки, состоящий из символов совокупностей BASIC LATIN и BASIC ARABIC, может быть определен следующим образом:

My-Character-String : : = BMPString (FROM (BasicLatin | BasicArabic))

Примечание — Приведенная конструкция необходима потому, что более простая.

My-Character-String : : = BMPString (BasicLatin | BasicArabic) допускает лишь строки, состоящие целиком из символов либо BASIC LATIN, либо BASIC ARABIC, но не из их смеси.

38 Канонический порядок символов

38.1 Для создания подтипов с помощью "ValueRange" и возможности использования правил кодирования определен канонический порядок символов для UniversalString, BMPString, NumericString, PrintableString, VisibleString и IA5String.

38.2 Для целей только данного раздела каждый символ имеет однозначное соответствие ячейке кодовой таблицы, независимо от того, присвоено ли ячейке имя символа и представление, является ли символ управляющим или печатным, комбинированным или некомбинированным.

38.3 Канонический порядок абстрактных символов определяется каноническим порядком их ячеек.

38.4 Для UniversalString канонический порядок ячеек определяется (см. ИСО/МЭК 10646-1) как:

256★(256★(128★(номер группы) + (номер плоскости)) + (номер строки)) + (номер ячейки)

Полный набор содержит ровно 128★256★256★256 символов. Конечные точки диапазонов "ValueRange" в нотациях "PermittedAlphabet" (или отдельных символов) могут быть заданы, используя либо ссылку на значение ACH.1, определенную в модуле ASN1-CHARACTER-MODULE, либо (когда графический символ является недвусмысленным в контексте спецификации) графическим символом в "cstring" (модуль ASN1-CHARACTER-MODULE определен в 37.1). Невозможно задать ячейку в качестве конечной точки диапазона или идентифицировать отдельный символ, когда этой ячейке не присвоено имя символа или представление.

38.5 Для BMPString канонический порядок ячеек определяется (см. ИСО/МЭК 10646-1) как

256★(номер строки) + (номер ячейки)

Полный набор содержит ровно 256★256 символов. Конечные точки диапазонов "ValueRange" в нотациях "PermittedAlphabet" (или отдельных символов) могут быть заданы, используя либо ссылку на значение ACH.1, определенную в модуле ASN1-CHARACTER-MODULE, либо (когда графический символ является недвусмысленным в контексте спецификации) графическим символом в "cstring". Невозможно задать ячейку в качестве конечной точки диапазона или идентифицировать отдельный символ, когда этой ячейке не присвоено имя символа или представление.

38.6 Для NumericString канонический порядок с возрастанием слева направо определяется (см. таблицу 4 в 36.2) как

(пробел) 0 1 2 3 4 5 6 7 8 9

Полный набор содержит ровно 11 символов. Конечная точка диапазона "ValueRange" (или отдельные символы) может быть задана использованием графического символа в "cstring".

Примечание — Этот порядок является тем же самым, что и порядок соответствующих символов в совокупности BASIC LATIN ИСО/МЭК 10646-1.

38.7 Для PrintableString канонический порядок с возрастанием слева направо определяется (см. таблицу 5 в 36.4) как

(пробел) (апостроф) (левая скобка) (правая скобка) (знак плюс)
(запятая) (дефис) (точка) (наклонная черта) 0123456789
(двоеточие) (знак равенства) (знак вопроса)
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Полный набор содержит ровно 74 символа. Конечная точка диапазона "ValueRange" (или отдельные символы) может быть задана использованием графического символа в "cstring".

Примечание — Этот порядок является тем же самым, что и порядок соответствующих символов в совокупности BASIC LATIN ИСО/МЭК 10646-1.

38.8 Для VisibleString канонический порядок ячеек определяется кодированием ИСО 646 (названным ISO 646 ENCODING) следующим образом:

(ISO 646 ENCODING) — 32

Примечание — Таким образом, канонический порядок тот же самый, что и для символов в ячейках 2/0—7/14 кодовой таблицы ИСО 646.

Полный набор содержит ровно 95 символов. Конечная точка диапазона "ValueRange" (или отдельные символы) могут быть заданы использованием графического символа в "cstring".

38.9 Для IA5String канонический порядок ячеек определяется кодированием ИСО/МЭК 646 следующим образом:

(ISO 646 ENCODING)

Полный набор содержит ровно 128 символов. Конечная точка диапазона "ValueRange" (или отдельные символы) может быть задана использованием графического символа в "cstring" или ссылкой на значение управляющего символа ИСО 646, определенной в 37.1.1.

39 Определение неограниченных типов символьных строк

В данном значении определяется тип, значениями которого являются значения любого символьного абстрактного синтаксиса. Абстрактный синтаксис может быть частью множества определенных контекстов в экземпляре соединения или может быть непосредственно указан для каждого экземпляра использования неограниченного типа символьной строки.

Примечания

1 Символьный абстрактный синтаксис (и один или несколько соответствующих символьных синтаксисов передачи) может быть определен любой организацией, имеющей право присваивать идентификаторы объектов АСН.1.

2 Профили, создаваемые сообществами по интересам, будут определять символьные абстрактные синтаксисы и синтаксисы передачи, которые должны поддерживаться для конкретных экземпляров или групп символьных строк. Будет принято включать ссылку на поддерживаемые синтаксисы в форму заявки о соответствии реализации протоколу. Группирование экземпляров для целей спецификации прикладного уровня может быть достигнуто использованием различных ссылок на типы АСН.1 (все из которых должны быть ссылками на тип CHARACTER STRING).

39.1 Неограниченный тип символьных строк (см. 3.8.69) должен указываться нотацией "UnrestrictedCharacterStringType"

UnrestrictedCharacterStringType : : = CHARACTER STRING

39.2 Этот тип имеет тег универсального класса 29.

39.3 Тип состоит из значений, представляющих:

а) значение символьной строки, которая может, но не обязательно, быть значением типа символьной строки АСН.1, или

б) идентификацию (по отдельности или вместе):

1) класса значений, содержащего это значение символьной строки (символьный абстрактный синтаксис), и

2) использованное кодирование (символьный синтаксис передачи) для отличия этого значения символьной строки от других значений в том же самом классе.

39.4 Неограниченный тип символьных строк имеет ассоциированный тип, который используется для обеспечения нотаций значения и подтипа для этого типа.

39.5 Ассоциированный тип для определения значения и подтипа, используя окружение автоматического тегирования, есть (с нормативными комментариями):

```
SEQUENCE {
    identification              CHOICE {
        syntaxes                    SEQUENCE {
            abstract                OBJECT IDENTIFIER,
            transfer                OBJECT IDENTIFIER }
    - - Идентификаторы объектов абстрактного синтаксиса и синтаксиса передачи - -,
        syntax                  OBJECT IDENTIFIER
    - - Идентификатор объекта для класса кодирования - -,
        presentation-context-id     INTEGER
    - - (Применяется только в среде ВОС)
    - - Согласованный контекст представления идентифицирует класс значения и его
    - - кодирование - - ,
        context-negotiation         SEQUENCE {
```

```
        presentation-context-id                 INTEGER,
        transfer-syntax                         OBJECT IDENTIFIER }
    - - (Применяется только в среде ВОС)
    - - Идет процесс согласования контекста для идентификации класса значения и его
    - - кодирования - -,
      transfer-syntax                   OBJECT IDENTIFIER
    - - Класс значения (например спецификация того, что оно является значением типа ACH.1)
    - - зафиксирован проектировщиком приложения (и, следовательно, известен как
    - - отправителю, так и получателю). Этот случай предназначен главным образом для
    - - поддержки выборочного-шифрования-полей (или других преобразований кодирования)
    - - типов ACH.1 - - ,
        fixed                   NULL
    - - Значение данных является значением фиксированного типа ACH.1 (и, следовательно,
    - - известно как отправителю, так и получателю) - - },
    data-value-descriptor       ObjectDescriptor OPTIONAL
    - - Обеспечивает человекочитаемую идентификацию класса значения - -,
    string-value                OCTET STRING}
  (WITH COMPONENTS {
    ... ,
    data-value-descriptor ABSENT})
```

Примечание — Неограниченный тип символьных строк не допускает включения значения "data-value-descriptor" вместе с "identification". Однако определение ассоциированного типа отражает базовую общность, которая существует между типом "встроенное-зпд", внешним типом и неограниченным типом символьных строк.

39.6 Нотация значения должна быть нотацией значения для ассоциированного типа, в которой значение "string-value" OCTET STRING представляет собой кодирование, использующее синтаксис передачи, специфицированный в "identification".

```
UnrestrictedCharacterStringValue : : = SequenceValue
        - - значение ассоциированного типа, определенного в 39.5
```

39.7 Пример неограниченного типа символьных строк приведен в С.2.8.

40 Нотация для типов, определенных в разделах 41—43

40.1 Нотацией для ссылки на типы, определенные в разделах 41—43, должна быть:

```
UsefulType : : = typereference
```

где ссылка "typereference" является одной из определенных в разделах 41—43 с использованием нотации ACH.1.

40.2 Теги типов "UsefulType" определены в разделах 41—43.

41 Обобщенное время

41.1 Этот тип должен указываться именем

```
GeneralizedTime
```

41.2 Тип состоит из значений, представляющих:

а) календарную дату, как определено в ИСО 8601, и

б) время дня с любой точностью, определенной в ИСО 8601, за исключением значения часов 24, которое не должно использоваться, и

в) местную поправку часов, как определено в ИСО 8601.

41.3 Тип определяется, используя ACH.1, следующим образом:

```
GeneralizedTime : : =
    [UNIVERSAL 24] IMPLICIT VisibleString
```

со значениями VisibleString, ограничивающимися строками символов, которые являются либо:

а) строкой, представляющей календарную дату, как определено в ИСО 8601, с четырехзначным представлением года, двузначными представлениями месяца и дня, без использования разделителей, с последующей строкой, представляющей время дня, как определено в ИСО 8601, без

разделителей, отличных от десятичной запятой и десятичной точки (как установлено в ИСО 8601), и без завершающего Z (как установлено в ИСО 8601), либо

б) символами из перечисления а) с последующей прописной буквой Z, либо

в) символами из перечисления а) с последующей строкой, представляющей местную поправку часов, как определено в ИСО 8601, без разделителей.

В случае а) время представляет местное время. В случае б) время представляет всемирное время. В случае в) часть строки, сформированная как в случае а), представляет местное время (t_1), а местная поправка часов (t_2) позволяет определить всемирное время следующим образом:

всемирное время равно $t_1 - t_2$

Примеры:

Случай а)

"19851106210627.3"

представляет местное время 21 ч 6 мин и 27,3 с 6 ноября 1985 г.

Случай б)

"19851106210627.3Z"

представляет указанное выше всемирное время.

Случай в)

"19851106210627.3—0500"

представляет местное время как в случае а) с отставанием на 5 ч от всемирного времени.

41.4 Тег определен в 41.3.

41.5 Нотацией значения должна быть нотация значения VisibleString, определенная в 41.3.

42 Всемирное время

42.1 Этот тип должен указываться именем

UTCTime

42.2 Тип состоит из значений, представляющих:

а) календарную дату,

б) время дня с точностью до минуты или секунды и

в) (факультативно) местную поправку часов.

42.3 Тип определяется, используя АСН.1, следующим образом:

UTCTime : : = [UNIVERSAL 23] IMPLICIT VisibleString

со значениями VisibleString, ограничивающимися строками следующих символов:

а) шесть цифр YYMMDD, где YY — две последние цифры года, MM — двузначное представление месяца (считая январь 01), DD — двузначное представление дня (от 01 до 31), и

б) либо

1) четыре цифры hhmm, где hh — часы (от 00 до 23), а mm — минуты (от 00 до 59), либо

2) шесть цифр hhmmss, где hh и mm — как в случае 1), а ss — секунды (от 00 до 59), и

в) либо

1) символа Z, либо

2) одного из символов "+" или "-" с последующими цифрами hhmm,

где hh — часы, а mm — минуты.

Альтернативы в случае б) позволяют варьировать точность представления времени.

В альтернативе в)1) время является всемирным. В альтернативе в)2) время (t_1), заданное в а) и б), является местным; поправка часов (t_2), заданная альтернативой в)2), позволяет определить всемирное время следующим образом:

всемирное время равно $t_1 - t_2$

Пример 1 — Если местное время - 7 ч 2 января 1982 г., а всемирное — 12 ч, то значением UTCTime является либо

"8201021200Z"

либо

"8201020700—0500"

Пример 2 — Если местное время — 7 ч 2 января 2001 г., а всемирное — 12 ч, то значением UTCTime является либо

"0101021200Z"

либо

"0101020700—0500"

42.4 Тег определен в 42.3.

42.5 Нотацией значения должна быть нотация значения VisibleString, определенная в 42.3.

43 Тип "описатель объекта"

43.1 Этот тип должен указываться именем

ObjectDescriptor

43.2 Тип состоит из человекочитаемых текстов, которые служат для описания объектов. Текст не является недвусмысленной идентификацией объекта, но подразумевается, что идентичный текст для разных объектов есть что-то необычное.

Примечание — Рекомендуется, чтобы уполномоченные по присвоению объектам значений типа "OBJECT IDENTIFIER" присваивали также значения типа "ObjectDescriptor" этим объектам.

43.3 Тип определяется, используя АСН.1, следующим образом:

ObjectDescriptor : : = [UNIVERSAL 7] IMPLICIT GraphicString

Строка "GraphicString" содержит текст, описывающий объект.

43.4 Тег определен в 43.3.

43.5 Нотацией значения должна быть нотация значения GraphicString, определенная в 43.3.

44 Ограниченные типы

44.1 Нотация "ConstrainedType" позволяет применять ограничение к (порождающему) типу, либо ограничивая его множество значений некоторым подтипом порождающего типа, либо (в типах "множество" или "последовательность") задавая, что отношения между компонентами применяются к значениям порождающего типа и значениям некоторого другого компонента в том же самом значении множества или последовательности. С ограничением может быть ассоциирован идентификатор исключения.

```
ConstrainedType : : =
    Type Constraint       |
    TypeWithConstraint
```

В первой альтернативе порождающий тип есть "Type", а ограничение задается "Constraint", как определено в 44.5. Вторая альтернатива определена в 44.4.

44.2 Когда нотация "Constraint" следует за нотацией типа "множество-из" или "последовательность-из", то она применяется к типу "Type" в (самой внутренней) нотации "множество-из" или "последовательность-из", а не к типу "множество-из" или "последовательность-из".

Примечание — Например следующее ограничение "(SIZE(1..64))" применяется к VisibleString, а не к SEQUENCE OF:

```
NamesOfMemberNations : : = SEQUENCE OF VisibleString (SIZE(1..64))
```

44.2.1 Когда нотация "Constraint" следует за нотацией селективного типа, она применяется к выборочному типу, а не к типу выбранной альтернативы.

Примечание — В следующем примере ограничение (WITH COMPONENTS {..., a ABSENT}) применяется к типу CHOICE, а не к выбранному типу SEQUENCE (см. 29.1 bis).

```
T : : =  CHOICE {
   a SEQUENCE {
       a INTEGER OPTIONAL,
       b BOOLEAN
   },
   b NULL
}
V : : = a < T  (WITH COMPONENTS {..., a ABSENT})
```

44.3 Когда нотация "Constraint" следует за нотацией "TaggedType", то интерпретация всей нотации одна и та же, независимо от того, рассматривается ли "TaggedType" или "Type" в качестве порождающего типа.

44.4 Как следствие интерпретации, установленной в 44.2, специальная нотация обеспечивается для того, чтобы ограничение применялось к типам "множество-из" или "последовательность-из". Это нотация "TypeWithConstraint":

```
TypeWithConstraint : : =
        SET Constraint OF  Type    |
        SET SizeConstraint OF  Type  |
        SEQUENCE Constraint OF  Type    |
        SEQUENCE SizeConstraint OF  Type
```

В первой и второй альтернативах порождающий тип есть "SET OF Type", а в третьей и четвертой – "SEQUENCE OF Type". В первой и третьей альтернативах ограничение есть "Constraint" (см. 44.5), а во второй и четвертой – "SizeConstraint" (см. 48.5).

Примечание — Хотя альтернативы "Constraint" включают соответствующие альтернативы "SizeConstraint", последние, не взятые в скобки, введены для обратной совместимости с ГОСТ Р ИСО/МЭК 8824.

44.5 Ограничение специфицируется нотацией "Constraint":

```
Constraint : : = "(" ConstraintSpec ExceptionSpec ")"
ConstraintSpec : : =
      SubtypeConstraint   |
      GeneralConstraint
```

Продукция "ExceptionSpec" определяется в разделе 45. Если она не используется вместе с маркером расширения (см. раздел 47), то должна присутствовать только в том случае, когда продукция "ConstraintSpec" включает в себя ссылку "DummyReference" (см. ИСО/МЭК 8824-4, 8.3) или является ограничением "UserDefinedConstraint" (см. ИСО/МЭК 8824-3, раздел 9).

44.6 Нотация "SubtypeConstraint" является нотацией общего назначения "ElementSetSpec" (раздел 46):

```
        SubtypeConstraint : : = ElementSetSpec
```

В данном контексте элементы являются значениями порождающего типа (управляющим множеством элементов является порождающий тип). В множестве должен быть по крайней мере один элемент.

45 Идентификатор исключения

45.1 В сложной спецификации АСН.1 имеется ряд мест, где существенно распознавать, что декодеры должны обрабатывать материал, который не полностью специфицирован. Такие случаи возникают, в частности, при использовании ограничения, которое определено с использованием параметра абстрактного синтаксиса (см. ИСО/МЭК 8824-4, раздел 10).

45.2 В таких случаях проектировщик приложения нуждается в идентификации действий, которые должны быть предприняты, когда нарушается некоторое зависящее от реализации ограничение. Идентификатор исключения обеспечивает недвусмысленный способ указания частей спецификации АСН.1, чтобы выделить действия, которые должны быть предприняты. Идентификатор состоит из символа "!" с последующим факультативным типом АСН.1 и значением этого типа. При отсутствии типа в качестве типа значения принимается INTEGER.

45.3 Если присутствует продукция "ExceptionSpec", то она указывает, что в стандарте имеется текст, говорящий о том, как обрабатывать нарушение ограничения, связанное с "!". Если она отсутствует, то реализаторы будут вынуждены либо идентифицировать текст, описывающий действия, которые необходимо предпринять, либо предпринимать зависящие от реализации действия, когда встретиться нарушение ограничения.

45.4 Нотация "ExceptionSpec" определяется следующим образом:

```
ExceptionSpec : : =  "!" ExceptionIdentification | empty
ExceptionIdentification : : = SignedNumber   |
                    DefinedValue                    |
                    Type ":" Value
```

Первые две альтернативы обозначают идентификаторы исключения целочисленного типа. Третья альтернатива обозначает идентификатор исключения ("Value") произвольного типа ("Type").

45.5 Когда для типа установлен ряд ограничений, несколько из которых имеют идентификаторы исключений, идентификатор исключения самого внешнего ограничения должен рассматриваться как идентификатор исключения этого типа.

45.6 когда для типов, используемых в арифметических установках, присутствует маркер расширения, идентификатор исключения игнорируется и не наследуется типом, ограниченным в результате арифметической установки.

46 Спецификация множества элементов

При осуществлении арифметической установки, включающей ограничения подтипа и множества значений, в этой установке используются только абстрактные значения, определенные корнем расширения. Все экземпляры нотации значения (включая ссылки на значения), используемые в этих ограничениях, обязательно относятся к абстрактному значению корня расширения. Если на самом внешнем уровне "ElementSetSpecs" нет маркера расширения, то результат арифметической установки не является расширяемым типом.

При осуществлении арифметической установки, включающей множества информационных объектов, все информационные объекты (а не только в корне расширения) используются в этой установке. Если какое-либо множество информационных объектов, относящееся к арифметической установке, является расширяемым или на самом внешнем уровне "ElementSetSpecs" есть маркер расширения, то результат арифметической установки является расширяемым типом.

46.1 В некоторых нотациях может быть специфицировано множество элементов некоторого идентифицированного класса элементов (управляющего). В таких случаях используется нотация "ElementSetSpecs":

```
ElementSetSpecs : : =
        RootElementSetSpec  |
        RootElementSetSpec "," "..."  |
        RootElementSetSpec "," "..." "," AdditionalElementSetSpec
RootElementSetSpec : : = ElementSetSpec
AdditionalElementSetSpec : : = ElementSetSpec
ElementSetSpec : : = Unions  |
        ALL Exclusions
Unions : : = Intersections  |
        UElems UnionMark Intersections
UElems : : = Unions
Intersections : : = IntersectionElements  |
        IElems IntersectionMark IntersectionElements
IElems : : = Intersections
IntersectionElements : : = Elements | Elems Exclusions
Elems : : = Elements
Exclusions : : = EXCEPT Elements
UnionMark : : = "|" | UNION
IntersectionMark : : = "^" | INTERSECTION
```

Примечания

1 Символ "^" и слово INTERSECTION — синонимы. Символ "|" и слово UNION — синонимы. Рекомендуется, чтобы во всей спецификации использовались либо символ, либо слово. И в том, и в другом случае может использоваться слово EXCEPT.

2 Старшинство операторов от старших к младшим следующее: "EXCEPT", "^","|". "ALL EXCEPT" специфицировано так, что оно не может чередоваться с другими ограничениями без использования скобок вокруг "ALL EXCEPT xxx" .

3 Всякий раз, когда встречается продукция "Elements", может появиться либо ограничение без скобок (например INTEGER (1..4)), либо ограничение подтипа в скобках (например INTEGER ((1..4)|9))).

4 Два оператора "EXCEPT" должны разделяться или "|", "^", "(", или ")", так что (A EXCEPT B EXCEPT C) не допускается; оно должно быть заменено на ((A EXCEPT B) EXCEPT C) или на (A EXCEPT (B EXCEPT C)).

5 Отметим, что ((A EXCEPT B) EXCEPT C) — то же самое, что и (A EXCEPT (B | C)).

6 Элементы, которые указываются "ElementSetSpecs", являются объединением элементов, указанных "RootElementSetSpec" и "AdditionalElementSetSpec".

46.2 Элементами, образующими множество, являются:

а) если в "ElementSetSpec" выбрана первая альтернатива, — то специфицированные в "Unions" [см. б)], в противном случае — все элементы управляющего, за исключением специфицированных в нотации "Elements" для "Exclusions";

б) если в "Unions" выбрана первая альтернатива, — то специфицированные в "Intersections" [см. в)], в противном случае — элементы, специфицированные по крайней мере один раз в "UElems" или в "Intersections";

в) если в "Intersections" выбрана первая альтернатива, — то специфицированные в "Intersection-Elements" [см. г)], в противном случае — элементы из специфицированных продукцией "IElems", которые так же специфицированы и продукцией "IntersectionElements";

г) если в "IntersectionElements" выбрана первая альтернатива, — то специфицированные в "Elements", в противном случае — элементы, которые специфицированы в "Elems", за исключением специфицированных в "Exclusions".

46.3 Нотация "Elements" определяется следующим образом:

```
Elements    : : =
      SubtypeElements    |
      ObjectSetElements  |
      "(" ElementSetSpec ")"
```

Элементами, специфицированными этой нотацией, являются:

а) определенные в разделе 48, если используется альтернатива "SubtypeElements". Эта нотация должна использоваться только в том случае, когда управляющий является типом, а фактически участвующий тип будет ограничиваться далее возможностями нотации. В этом контексте управляющий называется порождающим типом;

б) определенные в ГОСТ Р ИСО/МЭК 8824-2, 12.6, если используется нотация "Object-SetElements". Эта нотация должна использоваться только в том случае, когда управляющий является классом информационных объектов;

в) специфицированные нотацией "ElementSetSpec", если используется последняя альтернатива.

47 Маркер расширения

Примечание — Как и любая нотация ограничения, маркер расширения не влияет на одни правила кодирования АСН.1, такие как базовые правила кодирования, но влияет на другие, такие как упаковывающие правила кодирования.

47.1 Маркер расширения (многоточие) является указанием того, что ожидаются расширяющие дополнения. Он не делает каких-либо утверждений о том, как должны обрабатываться такие расширения, а только указывает, что при декодировании они не должны трактоваться как ошибка.

47.2 Совместное использование маркера расширения и идентификатора расширения является указанием, что ожидаются расширяющие дополнения, что они не должны при декодировании трактоваться как ошибка и что прикладные стандарты предписывают конкретные действия, которые должны быть предприняты приложением, если имеется нарушение ограничения. Рекомендуется применять эту нотацию в случаях, когда используется метод "запомнить и передать" или другая форма ретрансляции, указывая, что любые нераспознанные расширяющие дополнения должны возвращаться приложению для возможного перекодирования и ретрансляции.

47.3 Результат арифметической установки, включающей ограничения подтипа, множества значений или множества объектов, которые являются расширяемыми, описан в разделе 46.

47.4 Если в "ContainedSubtype" указан тип, определенный с расширяемым ограничением, то вновь определяемый тип не наследует маркер расширения и его расширяющие дополнения. Если вновь определяемый тип должен быть расширяемым, то маркер расширения должен быть добавлен к его "ElementSetSpecs" явным образом. Например:

```
A : : = INTEGER (0..10, . . . , 12)     - - A — расширяемый
B : : = INTEGER (A)              - - B — нерасширяемый и ограничен до (0—10)
C : : = INTEGER (A, . . .)       - - C – расширяемый и ограничен до (0– 10)
```

47.5 Если тип, определенный с расширяемым ограничением, ограничивается далее нотацией "ElementSetSpecs", которая не содержит маркер расширения, то ограничение получающегося типа

— нерасширяемое, а тип не наследует никаких расширяющих дополнений, которые могут присутствовать в порождающем типе. Например:

```
A : : = INTEGER (0..10, . . . )        - - A -- расширяемый
B : : = A (2..5)                       - - B — нерасширяемый
C : : = A                              - - C — расширяемый
```

47.6 Компоненты типов "множество", "последовательность" и "выбор", которые, согласно ограничению, должны отсутствовать, не могут присутствовать даже в том случае, когда тип "множество", "последовательность" или "выбор" является расширяемым.

Примечание — Внутренние ограничения типа не влияют на расширяемость.

Например:

```
A : : = SEQUENCE {
        a INTEGER
        b BOOLEAN OPTIONAL
        ...
}
B : : = A (WITH COMPONENTS {b ABSENT}) - - B — расширяемый, но 'b'
                                       - - не должен присутствовать ни в каком его значении.
```

47.7 Когда настоящий стандарт требует различия тегов (см. 24.5, 24.6, 26.3 и 28.2), то до проведения проверки на единственность тегов должно быть осуществлено следующее преобразование.

47.7.1 Новый элемент или альтернатива (называемый "концептуально добавляемым элементом", см. 47.7.2) концептуально добавляется в точке вставки расширения, если:

а) нет маркеров расширения, но расширяемость подразумевается заголовком модуля, и тогда добавляются маркер расширения и новый элемент как первое дополнение после маркера расширения, или

б) имеется единственный маркер расширения в CHOICE, SEQUENCE или SET, и тогда новый элемент добавляется в конце CHOICE, SEQUENCE или SET непосредственно перед закрывающей скобкой, или

в) имеется два маркера расширения в CHOICE, SEQUENCE или SET и тогда новый элемент добавляется непосредственно перед вторым маркером расширения.

47.7.2 Этот концептуально добавляемый элемент служит исключительно для проверки правильности применения правил, требующих различия тегов (см. 24.5, 24.6, 26.3 и 28.2). Он концептуально добавляется после применения автоматического тегирования (если оно применяется) и раскрытия COMPONENTS OF.

47.7.3 Концептуально добавляемый элемент определяется как имеющий тег, который отличен от тегов всех обычных типов АСН.1, но который согласуется с тегами всех концептуально добавляемых элементов и с неопределенным тегом открытого типа, как специфицировано в ГОСТ Р ИСО/МЭК 8824-2, 14.2, примечание 2.

Примечание — Правила, касающиеся единственности тегов относительно концептуально добавляемых элементов и открытого типа, вместе с правилами, требующими различия тегов (см. 24.5, 24.6, 26.3 и 28.2), являются необходимыми и достаточными, чтобы гарантировать, что:

а) любое неизвестное расширяющее дополнение может быть недвусмысленно приписано к единственной точке вставки при декодировании BER;

б) неизвестные расширяющие дополнения никогда не могут быть перепутаны с элементами OPTIONAL.

В PER эти правила достаточны, но не необходимы для гарантии указанных свойств. Тем не менее эти правила вводятся как правила АСН.1 для обеспечения независимости нотации от правил кодирования.

47.7.4 Если с этими концептуально добавленными элементами нарушаются правила, требующие различия тегов, то в спецификации была неправильно использована нотация расширения.

Примечание — Целью приведенных выше правил является установление точных ограничений, вытекающих из использования точек вставки (в частности, тех, которые не находятся в конце CHOICE, SEQUENCE или SET). Ограничения предназначены для того, чтобы гарантировать, что в BER, DER и CER можно неизвестный элемент, полученный системой версии 1, недвусмысленно приписать конкретной точке вставки. Это важно, когда обработка расширений таких добавленных элементов различна для разных точек вставки.

47.8 Примеры

47.8.1 Пример 1

```
A : : = SET {
    a  A,
    b  CHOICE {
       c  C,
       d  D,
       ...,
       ...
    }
}
```

Это допустимо потому, что нет двусмысленности, так как любой добавляемый материал должен быть частью "b".

47.8.2 Пример 2

```
A : : = SET {
    a  A,
    b  CHOICE {
       c  C,
       d  D,
       ...,
       ...
    },
    ...,
    e  E
}
```

Это недопустимо потому, что добавление может быть частью "b" или быть на внешнем уровне "A", и система версии 1 не сможет решить, где оно находится.

47.8.3 Пример 3

```
A : : = SET {
    a  A,
    b  CHOICE {
       c  C,
       ...
    },
    d  CHOICE {
       e  E
       ...
    }
}
```

Это недопустимо потому, что добавление может быть частью "b" или "d".

47.8.4 Могут быть построены и более сложные примеры с расширяемыми выборами в расширяемых выборах или с расширяемыми выборами в элементах последовательности, помеченных как OPTIONAL или DEFAULT, но приведенные выше правила необходимы и достаточны, чтобы гарантировать, что элемент, отсутствующий в версии 1, может быть недвусмысленно приписан системой версии 1 к ровно одной точке вставки.

48 Элементы подтипа

48.1 Общие положения

Для "SubtypeElements" обеспечивается ряд различных форм нотации. Они идентифицированы ниже, а их синтаксис и семантика определяются в последующих пунктах. В таблице 6 приведена сводка того, какие нотации к каким порождающим типам могут применяться.

Таблица 6 — Применимость множеств значений подтипов

Тип	Single Value	Contained Subtype	Value Range	Permitted Alphabet	SizeConstraint	TypeConstraint	Inner TypeConstraints
Битовая строка	Да	Да	Нет	Да	Нет	Нет	Нет
Булевский	Да	Да	Нет	Нет	Нет	Нет	Нет
Выборочный	Да	Да	Нет	Нет	Нет	Нет	Да
Встроенное-здп	Да	Нет	Нет	Нет	Нет	Нет	Да
Перечислимый	Да	Да	Нет	Нет	Нет	Нет	Нет
Внешний	Да	Нет	Нет	Нет	Нет	Нет	Да
Экземпляр-из	Да	Да	Нет	Нет	Нет	Нет	Да
Целочисленный	Да	Да	Да	Нет	Нет	Нет	Нет
Вырожденный	Да	Да	Нет	Нет	Нет	Нет	Нет
Поле класса объектов	Да	Да	Нет	Нет	Нет	Нет	Нет
Идентификатор объекта	Да	Да	Нет	Нет	Нет	Нет	Нет
Строка октетов	Да	Да	Нет	Да	Нет	Нет	Нет
Открытый	Нет	Нет	Нет	Нет	Нет	Да	Нет
Вещественный	Да	Да	Да	Нет	Нет	Нет	Да
Ограниченный символьных строк	Да	Да	Да*	Да	Да	Нет	Нет
Последовательность	Да	Да	Нет	Нет	Нет	Нет	Да
Последовательность-из	Да	Да	Нет	Да	Нет	Нет	Да
Множество	Да	Да	Нет	Нет	Нет	Нет	Да
Множество-из	Да	Да	Нет	Да	Нет	Нет	Да
Неограниченный символьных строк	Да	Нет	Нет	Да	Нет	Нет	Да

* Допустимо только в "PermittedAlphabet" для BMPString, IA5String, NumericString, PrintableString, VisibleString и UniversalString.

```
SubtypeElements : : =
        SingleValue                |
        ContainedSubtype           |
        ValueRange                 |
        PermittedAlphabet          |
        SizeConstraint             |
        TypeConstraint             |
        InnerTypeConstraints
```

48.2 Единственное значение

48.2.1 Нотация для единственного значения "SingleValue" должна быть:

SingleValue : : = Value

где "Value" является нотацией значения для порождающего типа.

48.2.2 Нотация "SingleValue" специфицирует единственное значение порождающего типа, заданное "Value".

48.3 Содержащийся подтип

48.3.1 Нотация для содержащегося подтипа "ContainedSubtype" должна быть:

ContainedSubtype : : = Includes Type

Includes : : = INCLUDES | empty

Альтернатива "empty" для продукции "Includes" не должна использоваться, когда "Type" в "ContainedSubtype" является нотацией для вырожденного типа.

48.3.2 Нотация "ContainedSubtype" специфицирует все значения в порождающем типе, которые имеются и в "Type". Требуется, чтобы тип "Type" был совместим с порождающим типом, как установлено в F.6.3.

48.4 Диапазон значений

48.4.1 Нотация для диапазона значений "ValueRange" должна быть:

ValueRange : : = LowerEndpoint ".." UpperEndpoint

48.4.2 Нотация "ValueRange" специфицирует все значения в диапазоне, который определяется заданием значений конечных точек диапазона. Эта нотация может применяться только для целочисленных и вещественных типов и некоторых ограниченных типов символьных строк PermittedAlphabet (только BMPString, IA5String, NumericString, PrintableString, VisibleString и UniversalString).

Примечание — При создании подтипов "PLUS-INFINITY" превышает все значения "NumericReal", а "MINUS-INFINITY" меньше всех значений "NumericReal".

48.4.3 Каждая конечная точка диапазона является либо закрытой (в таком случае эта конечная точка задана), либо открытой (в таком случае эта конечная точка не задана). Для открытой точки спецификация включает символ "меньше чем" ("<"):

LowerEndpoint : : = LowerEndValue | LowerEndValue "<"

UpperEndpoint : : = UpperEndValue | "<" UpperEndValue

48.4.4 Конечная точка может быть не задана, в таком случае диапазон простирается в этом направлении настолько, насколько допускает порождающий тип:

LowerEndValue : : = Value | MIN

UpperEndValue : : = Value | MAX

Примечание — Когда "ValueRange" используется как ограничение "PermittedAlphabet", "LowerEndValue" и "UpperEndValue" должны быть размером 1.

48.5 Ограничение размера

48.5.1 Нотация для ограничения размера "SizeConstraint" должна быть:

SizeConstraint : : = SIZE Constraint

48.5.2 Нотация "SizeConstraint" может применяться только к типам битовых строк, строк октетов, символьных строк, "множество-из" или "последовательность-из".

48.5.3 Продукция "Constraint" специфицирует допустимые целые значения для длины заданных значений и имеет вид любого ограничения, которое может применяться к следующему порождающему типу:

INTEGER (0 . . MAX)

Продукция "Constraint" должна использовать альтернативу "SubtypeConstraint" для "ConstraintSpec".

48.5.4 Единица измерения зависит от порождающего типа следующим образом:

Тип	Единица измерения
битовая строка	бит
строка октетов	октет
символьная строка	символ
множество-из	значение компонента
последовательность-из	значение компонента

Примечание — Подсчет количества символов в данном разделе для определения размера значения символьной строки следует четко отличать от подсчета октетов. Подсчет символов следует интерпретировать в соответствии с определением совокупности символов, используемой в типе, в частности, относительно ссылок на стандарты, таблицы или регистрационные номера в регистре, которые появляются в таком определении.

48.6 Ограничение типа

48.6.1 Нотация ограничения типа "TypeConstraint" должна быть:

TypeConstraint : : = Type

48.6.2 Эта нотация применяется только к нотации открытого типа и ограничивает открытый тип значениями "Type".

48.7 Допустимый алфавит

48.7.1 Нотация для допустимого алфавита "PermittedAlphabet" должна быть:

PermittedAlphabet . : = FROM Constraint

48.7.2 Нотация "PermittedAlphabet" специфицирует все значения, которые могут быть построены с использованием подалфавита порождающей строки. Эта нотация может применяться только для ограниченных типов символьных строк.

48.7.3 Ограничение "Constraint" является любым, которое может применяться для порождающего типа (см. таблицу 6), за исключением использующих альтернативу "SubtypeConstraint" для "ConstraintSpec". Подалфавит включает символы, появляющиеся в одном или нескольких значениях порождающего типа строки, которые допускаются ограничением "Constraint".

48.8 Внутренние подтипы

48.8.1 Нотация для внутренних подтипов "InnerTypeConstraints" должна быть:

```
InnerTypeConstraints : : =
    WITH COMPONENT SingleTypeConstraint    |
    WITH COMPONENTS MultipleTypeConstraints
```

48.8.2 Нотация "InnerTypeConstraints" специфицирует только те значения, которые удовлетворяют совокупности ограничений на присутствие, и/или значения компонентов порождающего типа. Значение порождающего типа не специфицировано, если оно не удовлетворяет всем ограничениям, явно выраженным или подразумеваемым (см. 48.8.6). Эта нотация может применяться к выборочному типу, типам "множество-из", "последовательность-из", "множество" или "последовательность".

Примечание — Нотация "InnerTypeConstraints", применяемая к типу "множество" или "последовательность", игнорируется преобразованием COMPONENTS OF (см. 24.4 и 26.2).

48.8.3 Для типов, которые определены в терминах единственного другого (внутреннего) типа ("множество-из" и "последовательность-из"), ограничение принимает вид спецификации значения подтипа. Нотацией для такого случая является "SingleTypeConstraint":

SingleTypeConstraint : : = Constraint

Ограничение "Constraint" определяет подтип единственного другого (внутреннего) типа. Значение порождающего типа специфицировано, если и только если каждое внутреннее значение относится к подтипу, полученному применением "Constraint" к внутреннему типу.

48.8.4 Для типов, которые определены в терминах нескольких других (внутренних) типов (выборочный, "множество" и "последовательность"), может быть несколько ограничений на эти внутренние типы. Нотацией для такого случая является "MultipleTypeConstraints":

```
MultipleTypeConstraints : : = FullSpecification | PartialSpecification
FullSpecification : : =  "{" TypeConstraints "}"
PartialSpecification : : = "{" ". . ." "," TypeConstraints "}"
TypeConstraints : : =
    NamedConstraint |
    NamedConstraint "," TypeConstraints
NamedConstraint : : =
        identifier ComponentConstraint
```

48.8.5 Продукция "TypeConstraints" содержит список ограничений на типы компонентов порождающего типа. Для типа "последовательность" ограничения должны появляться упорядочено. Внутренний тип, к которому применяется ограничение, идентифицируется с помощью его идентификатора. Для данного компонента должно быть не более одной продукции "NamedConstraint".

48.8.6 Нотация "MultipleTypeConstraints" включает в себя либо нотацию "FullSpecification", либо нотацию "PartialSpecification". Когда используется "FullSpecification", подразумевается присутствие ограничения "ABSENT" на все внутренние типы, которые могут быть ограничены тем, что будут отсутствовать (см. 48.8.9) и не перечислены явно. Когда используется "PartialSpecification", то нет подразумеваемых ограничений, и любой внутренний тип может быть опущен из списка.

48.8.7 Конкретный внутренний тип может быть ограничен в терминах его присутствия (в значениях порождающего типа), его значения, или и того, и другого. Нотацией является "ComponentConstraint":

ComponentConstraint : : = ValueConstraint PresenceConstraint

48.8.8 Ограничение на значение внутреннего типа выражается нотацией "ValueConstraint":

ValueConstraint : : = Constraint | empty

Ограничение удовлетворяется значением порождающего типа только в том случае, если внутреннее значение относится к подтипу, заданному ограничением "Constraint", применяемым к внутреннему типу.

48.8.9 Ограничение на присутствие внутреннего типа должно быть выражено нотацией "PresenceConstraint":

PresenceConstraint : : = PRESENT | ABSENT | OPTIONAL | empty

Смысл этих альтернатив и ситуации, в которых они допустимы, определены в 48.8.9.1 –48.8.9.3.

48.8.9.1 Если порождающий тип является последовательностью или множеством, то тип компонента, помеченный "OPTIONAL", может быть ограничен как "PRESENT" (и в этом случае ограничение удовлетворяется тогда и только тогда, когда соответствующее значение компонента присутствует) или как "ABSENT" (и в этом случае ограничение удовлетворяется тогда и только тогда, когда соответствующее значение компонента отсутствует), или как "OPTIONAL" (и в этом случае нет ограничений на присутствие соответствующего значения компонента).

48.8.9.2 Если порождающий тип является выборочным, то тип компонента может быть ограничен как "ABSENT" (и в этом случае ограничение удовлетворяется тогда и только тогда, когда соответствующий тип компонента не используется в значении) или как "PRESENT" (и в этом случае ограничение удовлетворяется тогда и только тогда, когда соответствующий тип компонента используется в значении); для данного типа не должно быть более одного ключевого слова "PRESENT" в продукции "MultipleTypeConstraints".

Примечание — Поясняющий пример см. в С.4.6.

48.8.9.3 Смысл пустой альтернативы для "PresenceConstraint" зависит от того, используется "FullSpecification" или "PartialSpecification":

а) в "FullSpecification" она эквивалентна ограничению "PRESENT" для компонентов множества или последовательности, помеченных "OPTIONAL", и не накладывает других ограничений в противном случае;

б) в "PartialSpecification" никаких ограничений не накладывается.

ПРИЛОЖЕНИЕ А
(обязательное)

Использование нотации АСН.1—90

А.1 Сроки действия

Термин АСН.1—90 используется для указания, что нотация определена в ГОСТ Р ИСО/МЭК 8824—93. Термин текущая нотация АСН.1 используется для нотации, определенной в настоящем стандарте.

На дату публикации настоящего стандарта продолжает действовать ГОСТ Р ИСО/МЭК 8824—93 и соответствующий ему международный стандарт ИСО/МЭК 8824—90. Действие последнего зависит от решения ИСО/МЭК/СТК1/ПК21.

Продолжение действия прежней спецификации дает пользователям время на замену характеристик (в частности, ANY и использование макронотации) нотации АСН.1—90 текущей нотацией АСН.1. (Это может быть сделано без изменений битов в строках).

А.2 Смешанное использование АСН.1—90 и текущей нотации АСН.1

Как в АСН.1—90, так и в текущей нотации АСН.1 специфицирована семантическая конструкция верхнего уровня, которой является модуль АСН.1. Пользователь АСН.1 создает совокупность модулей АСН.1 и может импортировать определения из других модулей АСН.1.

Для любого данного модуля требуется, чтобы используемая нотация (полностью) соответствовала либо АСН.1—90, либо текущей нотации АСН.1, и пользователь спецификации должен ясно идентифицировать (указанием соответствующего стандарта), какая нотация используется для каждого модуля, текстуально включенного в спецификацию пользователя.

Может случиться так, что пользователь захочет модифицировать часть модуля с использованием новой нотации, но оставить другие части в старой нотации. Это может быть достигнуто (только) путем расщепления модуля на два других модуля.

Когда модуль соответствует нотации АСН.1—90, ссылки на типы и значения могут быть импортированы из модуля, определенного с использованием текущей нотации. Такие типы и значения должны быть ассоциированы с типами, которые могут быть определены с использованием нотации АСН.1—90. Например модуль, написанный с использованием нотации АСН.1—90, не может импортировать значение типа UniversalString, так как этот тип определен в текущей нотации, но не в АСН.1—90; однако он может импортировать значения, типы которых, например, INTEGER, IA5String и т. п..

Когда модуль соответствует текущей нотации АСН.1, ссылки на типы и значения могут быть импортированы из модуля, определенного с использованием нотации АСН.1—90. Макронотация АСН.1 не может быть импортирована. Нотация значения для импортированного типа должна использоваться в импортирующем модуле, только если присутствуют идентификаторы для значений SET, SEQUENCE и CHOICE, использованных в значении нотации, и если в значении нотации не требуется значение типа ANY. Ограничение внутреннего типа не должно применяться к импортированному типу, если компонент, который должен быть ограничен, не имеет идентификатора.

А.3 Переход к текущей нотации АСН.1

При модификации модуля (первоначально написанного в согласно нотации АСН.1—90) для соответствия текущей нотации следует учитывать следующие моменты.

а) Всем компонентам SET, SEQUENCE и CHOICE должны быть даны идентификаторы, не двусмысленные в данном экземпляре SET, SEQUENCE и CHOICE, и такие же идентификаторы должны быть включены в нотацию значения.

Примечание 1 — Значение нотации для типа CHOICE содержит двоеточие (":").

б) Все использования ANY и ANY DEFINED BY должны быть обеспечены подходящими определениями классов информационных объектов с заменой ANY и ANY DEFINED BY (и указанных компонентов) соответствующими ссылками на поля этого класса объектов. В большинстве случаев спецификация может быть улучшена за счет удачной вставки табличных ограничений и ограничений отношений компонентов. Во многих случаях спецификации может быть еще более улучшена, если табличное ограничение или ограничение отношений компонентов осуществляется как параметр типа.

в) Макроопределения должны быть заменены определением класса информационных объектов, параметризованным типом или параметризованным значением. Если раздел WITH SYNTAX удачно спроектирован в определении класса информационных объектов, то нотация, используемая для определения объектов этого класса, может быть сделана очень похожей на нотацию, определенную старым использованием макронотации.

г) Все экземпляры использования макронотации должны быть заменены либо эквивалентными определениями информационных объектов, либо ссылками на эквивалентные типы "ObjectClassFieldType", параметризованные типы или параметризованные значения. В большинстве случаев спецификация информационных объектов может быть существенно улучшена группировкой таких определений в множества информационных

объектов и ясным указанием, является ли обязательной поддержка всех информационных объектов множества, должны ли принимающие реализации приспосабливаться к зависящим от реализации расширениям этого множества информационных объектов и, если это так, как они должны обрабатывать полученные "неизвестные" значения. Может оказаться желательным рассмотреть возможность, что в последующей версии спецификации пользователя может быть расширено множество информационных объектов и дано руководство нынешним разработчикам, как такие расширения должны трактоваться.

д) Все появления EXTERNAL должны быть тщательно рассмотрены; хотя эта нотация остается допустимой в текущей нотации АСН.1, спецификация пользователя может быть улучшена следующим образом:

1) Рассмотрите использование нотации INSTANCE OF (предпочтительно с табличным ограничением в качестве параметра типа, как обсуждалось выше для ANY и ANY DEFINED BY) вместо нотации EXTERNAL; во многих случаях это не изменит битов в строке.

2) Когда сохраняется EXTERNAL, использование внутренних подтипов ассоциированного типа (см. 33.5) может помочь придать точность спецификации тому, используются ли идентификаторы контекстов представления, или это не допускается. Здесь же применимы предшествующие комментарии (см. раздел 33), которые дают руководство о том, какие значения EXTERNAL должны поддерживаться и что должны делать реализации, если получены неподдерживаемые значения.

3) Рассмотрите замену

CHOICE { external EXTERNAL, embedded-pdv EMBEDDED PDV }

(с использованием внутренних подтипов, если это подходит) для того, чтобы обеспечить постепенный переход распределенных приложений к текущей нотации. Это может повлиять на биты в строке и обычно может быть сделано как часть изменения версии протокола. Использование EMBEDDED PDV (практически для новой спецификации) обычно дает большую гибкость, как можно увидеть из сравнения ассоциированных типов; более того, всеми правилами кодирования, определенными в ИСО/МЭК 8825-1, EMBEDDED PDV кодируется более эффективно, чем EXTERNAL.

е) Может оказаться возможным улучшить удобочитаемость нотации в существующих модулях АСН.1 (без изменения битов в строке) вставкой AUTOMATIG TAGS в заголовок модуля и удалением некоторых или всех тегов.

Примечание 2 — Это следует делать с осторожностью и пониманием метода действия автоматического тегирования, так как, если этот прием использовать некорректно, биты в строке изменятся.

ж) Если AUTOMATIG TAGS не применяется в существующих модулях так, как описано в е), то нежелательно добавлять определения новых типов в существующий модуль, а лучше создать новый модуль (с автоматическим тегированием) для определений новых типов. Это позволяет использовать преимущества автоматического тегирования без изменения битов в строке.

и) Следует уделить внимание полям, содержащим символьные строки, и посмотреть, нельзя ли задействовать нотации CHARACTER STRING, BMPString или UniversalString. Однако обычно это изменит биты в строке и может быть проведено как часть изменения версии.

к) Необходимо добавить идентификаторы "mantissa", "base" "exponent" ко всем нотациям вещественных значений, которые используют альтернативу "NumericRealValue" в продукции "RealValue". Рассмотрение должно ограничиваться значениями "base" 2 и 10 в нотации типа.

В общем случае возможно существенное улучшение удобочитаемости, эффективности, точности и гибкости за счет использования новой нотации АСН.1 (в частности, если использовать все преимущества табличных ограничений, ограничений связи компонентов и параметризации, а также новые типы символьных строк). Всем пользователям АСН.1—90 необходимо осуществить переход на новую нотацию АСН.1, либо при пересмотре своих спецификаций, либо как самостоятельное действие, если такой пересмотр не предвидется.

В общем случае, дополнение существующих модулей с использованием нотации, не соответствующей текущей нотации АСН.1, рассматривается как ошибка, даже если ссылка на спецификации АСН.1—90 сохраняется в таких модулях. В частности, запрещается использование макронотации, ANY, ANY DEFINED BY, а также новых конструкций SET, SEQUENCE и CHOICE без недвусмысленных идентификаторов.

ПРИЛОЖЕНИЕ В
(обязательное)

Присвоение значений идентификаторов объектов

В настоящем стандарте присвоены следующие значения:

Раздел 36.3 Значение идентификатора объекта
{ joint-iso-itu-t asn1 (1) specification (0) characterStrings (1) numericString (0) }
Значение описателя объекта
"NumericString ASN.1 type"

Раздел 36.5 Значение идентификатора объекта
{ joint-iso-itu-t asn1 (1) specification (0) characterStrings (1) printableString (1) }
Значение описателя объекта
"PrintableString ASN.1 type"

Раздел 36.1 Значение идентификатора объекта
{ joint-iso-itu-t asn1 (1) specification (0) modules (0) iso10646 (0) }
Значение описателя объекта
"ASN1 character Module"

ПРИЛОЖЕНИЕ С
(справочное)

Примеры и указания

Настоящее приложение содержит примеры использования АСН.1 при описании (гипотетических) структур данных. Оно так же содержит указания, или руководства по использованию различных характеристик АСН.1. Если не оговорено противное, то принимается окружение автоматического тегирования AUTOMATIG TAGS.

С.1 Пример персональной записи

Использование АСН.1 иллюстрируется на примере простейшей гипотетической персональной записи.

С.1.1 Неформальное описание персональной записи

Структура персональной записи и значения для конкретного лица показаны ниже.

Name (имя):	John P Smith
Title (должность):	Director
Employee Number (табельный номер):	51
Date of Hire (дата приема на работу):	17 сентября 1971
Name of Spouse (имя супруги):	Mary T Smith
Number of Children (число детей):	2
Child Information (информация о детях)	
Name (имя):	Ralph T Smith
Date of Birth (дата рождения):	11 ноября 1957
Child Information (информация о детях)	
Name (имя):	Susan B Jones
Date of Birth (дата рождения):	17 июля 1959

С.1.2 Описание АСН.1 структуры записи

Структура каждой персональной записи ниже описана формально с использованием стандартной нотации для типов данных:

```
PersonnelRecord : : = [APPLICATION 0] SET
  { name            Name,
    title           VisibleString,
    number          EmployeeNumber,
    dateOfHire      Date,
    nameOfSpouse    Name,
    children        SEQUENCE OF ChildInformation DEFAULT {}
  }
ChildInformation : : = SET
  { name            Name,
    dateOfBirth     Date,
  }
Name : : = [APLICATION 1] SEQUENCE
  { givenName       VisibleString,
    initial         VisibleString,
    familyName      VisibleString,
  }
EmployeeNumber : : = [APLICATION 2] INTEGER
Date : : = [APLICATION 3] VisibleString
```

Данный пример иллюстрирует аспекты синтаксического анализа АСН.1. Синтаксическая конструкция "DEFAULT" может применяться только для компонента "SEQUENCE" или "SET", но не может применяться к элементу "SEQUENCE OF". Таким образом, "DEFAULT {}" в "PersonnelRecord" применяется к "children", а не к "ChildInformation".

С.1.3 Описание АСН.1 значения записи

Значение для персональной записи Джона Смита ниже описано формально с использованием стандартной нотации для значений данных:

```
{ name              {givenName "John", initial "P", familyName "Smith"},
  title             "Director",
  number            51,
  dateOfHire        "19710917",
  nameOfSpouse      {givenName "Mary", initial "T", familyName "Smith"},
  children
```

```
{ {name {givenName "Ralph", initial "T", familyName "Smith"},
   dateOfBirth "19571111"},
   {name {givenName "Susan", initial "B", familyName "Jones"},
   dateOfBirth "19590717"}
}
```

С.2 Руководство по использованию нотации

Типы данных и формальная нотация, определенные в настоящем стандарте, являются гибкими и позволяют проектировать широкий диапазон протоколов. Однако эта гибкость иногда может приводить к путанице, особенно когда нотация используется впервые. Данное приложение является попыткой минимизировать возможную путаницу и дает руководство (и примеры) использования нотации. Для каждого встроенного типа приводится одно или несколько указаний по его использованию. Типы символьных строк (например VisibleString) и типы, определенные в разделах 41—43, здесь не рассматриваются.

С.2.1 Булевский тип

С.2.1.1 Булевский тип используется для моделирования логических значений, например ответов да-нет на поставленные вопросы.

Пример

```
Employed : : = BOOLEAN
```

С.2.1.2 При назначении ссылочного имени булевскому типу выбирается то имя, которое описывает состояние "истинно".

Пример

```
Married : : = BOOLEAN            - - женат
```

но не

```
MaritalStatus : : = BOOLEAN            - - семейное положение
```

С.2.2 Целочисленный тип

С.2.2.1 Целочисленный тип используется для моделирования значений (практически не ограниченных) кардинальных или целочисленных переменных.

Пример

```
CheckingAccountBalance : : = INTEGER - - баланс в центах,
                  - - отрицательное значение означает перерасход
balance CheckingAccountBalance : : = 0
```

С.2.2.2 Максимальное и минимальное допустимые значения целочисленного типа определяются как поименованные числа.

Пример

```
DayOfTheMonth  : : = INTEGER {first (1), last (31)}
today DayOfTheMonth  : : = first
unknown DayOfTheMonth  : : = 0
```

Поименованные числа "first" и "last" выбраны из-за их семантического значения, но не исключают возможности значений DayOfTheMonth меньших 1, больших 31 или в интервале 1—31.

Для того чтобы ограничить значения DayOfTheMonth только лишь значениями "first" и "last" следует писать:

```
DayOfTheMonth  : : = INTEGER {first (1), last (31)} (first | last)
```

Для того чтобы ограничить значения DayOfTheMonth значениями от 1 до 31, следует писать:

```
DayOfTheMonth  : : = INTEGER {first (1), last (31)} (first . . last)
dayOfTheMonth DayOfTheMonth  : : =  4
```

С.2.3 Перечислимый тип

С.2.3.1 Перечислимый тип используется для моделирования значений переменных с тремя и более состояниями. Значения присваиваются, начиная с нуля, если единственным ограничением является их отличие друг от друга.

Пример

```
DayOfTheWeek  : : = ENUMERATED{sunday (0), monday (1), tuesday (2),
                    wednesday (3), thursday (4), friday (5), saturday (6)}
- - Дни недели: воскресенье (0), понедельник (1), вторник (2), . . .
firstDayOfTheWeek : : = sunday
```

Хотя перечисления "sunday", "monday" и т. д. выбраны из-за их семантического смысла, DayOfTheWeek ограничивается только этими значениями. Более того, могут быть присвоены только значения "sunday", "monday" и т. д., а эквивалентные целочисленные значения недопустимы.

С.2.3.2 Расширяемый перечислимый тип используется для моделирования значений переменной, которая в настоящее время имеет два состояния, но может иметь дополнительные состояния в последующих версиях протокола.

Пример

```
MaritalStatus : : = ENUMERATED {single, married}
        - - Первая версия записи
        - - семейного положения: одинокий, женатый
```

в предвидении

```
MaritalStatus : : = ENUMERATED{single, married, . . . , widowed}
        - - Вторая версия записи
        - - семейного положения: одинокий, женатый, вдовый
```

и далее

```
MaritalStatus : : = ENUMERATED{single, married, . . . , widowed, divorced}
        - - Третья версия записи
        - - семейного положения: одинокий, женатый, вдовый, разведенный
```

С.2.4 Действительный тип

С.2.4.1 Действительный тип используется для моделирования приближенных чисел.

Пример

```
AngleInRadians : : = REAL   - - угол в радианах
pi REAL : : = {mantissa 31415926535897932384626433832 79, base 10, exponent —30}
```

С.2.4.2 Проектировщики приложений могут захотеть гарантировать полную совместимость действительных значений, независимо от различия в техническом представлении плавающей точки, и использование в реализациях (например) представления с простой или двойной точностью. Это может быть достигнуто следующим образом:

```
App-X-Real : : = REAL (WITH COMPONENTS {
                    mantissa (—16777215 . . 16777215),
                    base (2),
                    exponent (—125 . . 128)})
- - Отправители не должны передавать значения вне этих диапазонов, а соответствующие получатели
- - должны быть способны принимать и обрабатывать значения из этих диапазонов.
girth App-X-Real : : = {mantissa 16, base 2, exponent 1}
```

С.2.5 Битовая строка

С.2.5.1 Битовая строка используется для моделирования двоичных данных, формат и длина которых не заданы или заданы где-либо в другом месте, а длина в битах не обязательно кратна восьми.

Пример

```
G3FacsimilePage : : = BIT STRING
- - последовательность битов, соответствующая Рекомендации МККТТ Т. 4
image G3FacsimilePage : : = '10011010010000111010'B
trailer BIT STRING : : = '0123456789ABCDEF'H
body1 G3FacsimilePage : : = '1101'B
body2 G3FacsimilePage : : = '1101000'B
```

Нотации "body1" и "body2" являются разными абстрактными значениями, так как завершающие нулевые биты являются значащими (так как отсутствует конструкция "NamedBitList" в определении G3FacsimilePage).

С.2.5.2 Битовая строка с ограничением размера используется для моделирования значений битовых полей фиксированного размера.

Пример

```
BitField : : = BIT STRING (SIZE (12))
map1 BitField : : = '100110100100'B
map2 BitField : : = '9A4'H
map3 BitField : : = '1001101001'B
                    - - Недопустимо — нарушено ограничение размера
```

Нотации "map1" и "map2" являются одинаковыми абстрактными значениями, так как завершающие четыре нулевых бита в "map2" не являются значащими.

С.2.5.3 Битовая строка используется для моделирования значений типа bit map — упорядоченной совокупности логических переменных, указывающей, выполнено ли конкретное условие для каждого соответствующего объекта упорядоченной совокупности объектов.

```
DaysOfTheWeek : : = BIT STRING {
                    sunday (0), monday (1), tuesday (2),
                    wednesday (3), thursday (4), friday (5),
                    saturday (6)} (SIZE (0 . . 7))
    - - Дни недели: воскресенье (0), понедельник (1), вторник (2), . . .
sunnyDaysLastWeek1 DaysOfTheWeek : : = {sunday, monday, wednesday}
    - - Солнечные дни на последней неделе:
    - - воскресенье, понедельник, среда
sunnyDaysLastWeek2 DaysOfTheWeek : : = '1101'B
```

```
sunnyDaysLastWeek3 DaysOfTheWeek : : = '1101000'B
sunnyDaysLastWeek4 DaysOfTheWeek : : = '11010000'B
                        - - Недопустимо — нарушено ограничение размера
```

Если значение битовой строки короче 7 бит, то отсутствующие биты указывают пасмурный день, следовательно первые три приведенные нотации имеют одно и то же абстрактное значение.

С.2.5.4 Битовая строка используется для моделирования значений типа bit map — упорядоченной, фиксированного размера совокупности логических переменных, указывающей, выполнено ли конкретное условие для каждого соответствующего объекта упорядоченной совокупности объектов.

```
DaysOfTheWeek : : = BIT STRING {
                        sunday (0), monday (1), tuesday (2),
                        wednesday (3), thursday (4), friday (5),
                        saturday (6)} (SIZE (7))
    - - Дни недели: воскресенье (0), понедельник (1), вторник (2), . . .
sunnyDaysLastWeek1 DaysOfTheWeek : : = {sunday, monday, wednesday}
    - - Солнечные дни на последней неделе:
    - - воскресенье, понедельник, среда
sunnyDaysLastWeek2 DaysOfTheWeek : : = '1101'B
                        - - Недопустимо — нарушено ограничение размера
sunnyDaysLastWeek3 DaysOfTheWeek : : = '1101000'B
sunnyDaysLastWeek4 DaysOfTheWeek : : = '11010000'B
                        - - Недопустимо — нарушено ограничение размера
```

Первая и третья нотации имеют одно и то же абстрактное значение.

С.2.5.5 Битовая строка с поименованными битами используется для моделирования значений совокупности связанных логических переменных.

Пример

```
PersonalStatus : : = BIT STRING
        {married (0), employed (1), veteran (2), collegeGraduate (3)}
- - Личный статус: женат, работающий, ветеран, выпускник колледжа
billClinton PersonalStatus : : = (married, employed, collegeGraduate)
hillaryClinton PersonalStatus : : = '110100'B
```

Нотации "billClinton" и "hillaryClinton" имеют одно и то же абстрактное значение.

С.2.6 Строка октетов

С.2.6.1 Строка октетов используется для моделирования двоичных данных, формат и длина которых не заданы или заданы где-либо в другом месте, а длина в битах кратна восьми.

Пример

```
G4FacsimileImage : : = OCTET STRING
- - последовательность октетов,
- - соответствующая Рекомендациям МККТТ Т. 5 и Г. 6
  image G4FacsimilePage : : = '3FE2EBAD471005'H
```

С.2.6.2 Использование ограниченной символьной строки предпочтительнее строки октетов, когда обе из них приемлемы.

Пример

```
Surname : : = PrintableString
president Surname : : = "Clinton"
```

С.2.7 Строки UniversalString и BMPString

Тип BMPString используется для моделирования любых строк информации, полностью состоящих из символов базовой многоязычной плоскости (BMP) ИСО/МЭК 10646-1, а тип UniversalString — для моделирования строк, состоящих из символов ИСО/МЭК 10646-1, не входящих в BMP.

С.2.7.1 Для обозначения уровня реализации, накладывающего ограничения на использование комбинированных символов, используются обозначения "Level1" и "Level2".

Пример

```
RussianName : : = Cyrillic (Level1)
        - - В RussianName не используются комбинированные символы
SaudiName : : = BasicArabic (SIZE (1 . . . 100)^Level2)
- - В SaudiName используется подмножество комбинированных символов
```

С.2.7.2 Совокупность может быть расширена до выбранного подмножества путем использования нотации "UnionMark" (см. раздел 44).

Пример

```
KatakanaAndBasicLatin : : = UniversalString (FROM(Katakana | BasicLatin))
```

С.2.8 Тип CHARACTER STRING

Неограниченный тип символьных строк используется для моделирования любых строк информации, которые не могут быть смоделированы ни одним из ограниченных типов символьных строк. Необходимо гарантировать спецификацию репертуара символов и их кодирования в октетах.

Пример

```
PackedBCDString : : = CHARACTER STRING (WITH COMPONENTS{ identification (WITH COMPONENTS{
                                                                    fixed PRESENT})
    - - Абстрактным синтаксисом и синтаксисом передачи должны быть
    - - определяемые ниже  packedBCDStringAbstractSyntax
    - - и packedBCDStringTransferSyntax, соответственно
          })
- - Значение идентификатора объекта для символьного абстрактного синтаксиса (набора символов),
- - алфавитом которого являются цифры от 0 до 9
packedBCDStringAbstractSyntaxId OBJECT IDENTIFIER : : =
    {joint-iso-itu-t xxx (999) yyy (999) zzz (999) packedBCD (999) charSet (0)}
- - Значение идентификатора объекта для символьного синтаксиса передачи, который упаковывает по две
- - цифры в октет; цифры кодируются от 0000 до 1001; 1111 используется для заполнения октета.
packedBCDStringTransferSyntaxId OBJECT IDENTIFIER : : =
    {joint-iso-itu-t xxx (999) yyy (999) zzz (999) packedBCD (999) characterTransferSyntax (1)}
- - Кодирование packedBCDString будет содержать только определенное кодирование символов, с любой
- - необходимой длиной поля, а в случае BER — с передающим тег полем. Значения идентификаторов
объектов не передаются, так как должно быть задано "fixed".
```

Примечание — По правилам кодирования не обязательно кодировать значения типа CHARACTER STRING в форме, которая всегда включает в себя значения идентификаторов объектов, хотя они гарантируют, что абстрактное значение сохраняется при кодировании.

С.2.9 Вырожденный тип

Вырожденный тип null используется для указания отсутствия компонента последовательности.

Пример

```
PatientIdentifier : : = SEQUENCE {
      name                VisibleString,
      roomNumber          CHOICE {
      room                INTEGER,
      outPatient          NULL - - если пациент выписан - -
   }
}
lastPatient PatientIdentifier : : = {
      name                "Jane Doe",
      roomNumber          outPatient : NULL
}
```

С.2.10 Последовательность и последовательность-из

С.2.10.1 Тип "последовательность-из" используется для моделирования совокупности переменных, тип которых один и тот же, количество велико или непредсказуемо, а порядок — существенен.

Пример

```
NamesOfMemberNations : : = SEQUENCE OF VisibleString
- - в алфавитном порядке
firstTwo NamesOfMemberNations : : = {"Australia", "Austria"}
```

С.2.10.2 Тип "последовательность" используется для моделирования совокупности переменных, тип которых один и тот же, количество известно и невелико, а порядок — существенен, при условии, что введенная разметка совокупности вряд ли изменится в последующих версиях протокола.

Пример

```
NamesOfOfficers : : = SEQUENCE {
      president           VisibleString,
      vicePresident       VisibleString,
      secretary           VisibleString}
      - - Перечень официальных лиц: президент, вицепрезидент, секретарь
acmeCorp NamesOfOfficers : : =
      president            "Jane Doe",
      vicePresident        "John Doe",
      secretary            "Joe Doe"}
```

С.2.10.3 Тип "последовательность" используется для моделирования совокупности переменных, типы которых различны, количество известно и невелико, а порядок — существенен, при условии, что введенная разметка совокупности вряд ли изменится в последующих версиях протокола.

Пример

```
Credentials : : = SEQUENCE {
        userName            VisibleString,
        password            VisibleString,
        accountNumber       INTEGER}
```

С.2.10.4 Расширяемый тип "последовательность" используется для моделирования совокупности переменных, порядок которых существенен, количество их в настоящий момент известно и невелико, но ожидается, что оно должно возрасти.

Пример

```
Record : : = Sequence {        - - Первая версия Record
    userName            VisibleString,
    password            VisibleString,
    accountNumber       INTEGER,
    ...,
    ...
    }
```

в предвидении

```
Record : : = Sequence {            - - Вторая версия Record
    userName            VisibleString,
    password            VisibleString,
    accountNumber       INTEGER,
    ...,
    [[              - - Расширяющее дополнение к второй версии
      lastLoggedIn                  GeneralizedTime OPTIONAL,
      minutesLastLoggedIn           INTEGER
    ]],
    ...
    }
```

и далее

```
Record : : = Sequence {            - - Третья версия Record
    userName            VisibleString,
    password            VisibleString,
    accountNumber       INTEGER,
    ...,
    [[              - - Расширяющее дополнение к второй версии
      lastLoggedIn                  GeneralizedTime OPTIONAL,
      minutesLastLoggedIn           INTEGER
    ]],
    [[              - - Расширяющее дополнение к третьей версии
      certificate                   Certificate,
      thumb                         ThumbPrint OPTIONAL
    ]],
    ...
    }
```

С.2.11 Множество и множество-из

С.2.11.1 Тип "множество" используется для моделирования совокупности переменных, количество которых известно и невелико, а порядок — не существенен. Если не используется автоматическое тегирование, то каждая переменная идентифицируется контекстно зависимым тегированием, как показано ниже. (При автоматическом тегировании теги необязательны.)

Пример

```
UserName : : = SET {
    personalName        [0] VisibleString,
    organizationName    [1] VisibleString,
    countryName         [2] VisibleString }
- - Имя пользователя: личное имя, название организации, название страны
user UserName : : =  {
    countryName         "Nigeria",
    personalName        "Jonas Maruba",
    organizationName    "Meteorology, Ltd."}
```

С.2.11.2 Тип "множество" с ключевым словом "OPTIONAL" используется для моделирования совокупности переменных, являющейся (собственным или несобственным) подмножеством другой совокупности переменных, количество которых известно и разумно мало, а порядок — не существенен. Если не используется

автоматическое тегирование, то каждая переменная идентифицируется контекстно зависимым тегированием, как показано ниже. (При автоматическом тегировании теги необязательны).

```
UserName : : = SET {
    personalName            [0] VisibleString,
    organizationName        [1] VisibleString OPTIONAL
                    - - по умолчанию — местная организация - - ,
    countryName             [2] VisibleString OPTIONAL
                    - - по умолчанию — данная страна - - }
```

С.2.11.3 Расширяемый тип "множество" используется для моделирования совокупности переменных, разметка которой, вероятно, изменится в последующих версиях протокола. В следующем примере принято, что в определении модуля задано AUTOMATIC TAGS.

Пример

```
UserName : : = SET {        - - Первая версия UserName
    personalName            VisibleString,
    organizationName        VisibleString OPTIONAL,
    countryName             VisibleString OPTIONAL,
    ...,
    ...
}
user UserName : : = {personalName "Jonas Maruba"}
```

в предвидении

```
UserName : : = SET {        - - Вторая версия UserName
    personalName            VisibleString,
    organizationName        VisibleString OPTIONAL,
    countryName             VisibleString OPTIONAL,
    ...,
    [[          - - Расширяющее дополнение к второй версии
      intenetEmailAddress   VisibleString,
      faxNumber             VisibleString OPTIONAL
    ]],
    ...
}
user UserName : : = {
    personalName            "Jonas Maruba",
    intenetEmailAddress     "jonas@meteor.ngo.com"
    }
```

и позже

```
UserName : : = SET {        - - Третья версия UserName
    personalName            VisibleString,
    organizationName        VisibleString OPTIONAL,
    countryName             VisibleString OPTIONAL,
    ...,
    [[          - - Расширяющее дополнение к второй версии
      intenetEmailAddress   VisibleString,
      faxNumber             VisibleString OPTIONAL
    ]],

    phoneNumber             VisibleString OPTIONAL,
                - - Расширяющее дополнение к третьей версии
    ...
}
user UserName : : = {
    personalName            "Jonas Maruba",
    intenetEmailAddress     "jonas@meteor.ngo.com"
    }
```

С.2.11.4 Тип "множество-из" используется для моделирования совокупности переменных, тип которых один и тот же, а порядок — не существенен.

Пример

```
Keywords : : = SET OF VisibleString - - в произвольном порядке
someASN1Keywords Keywords : : = {"INTEGER", "ROOLEAN", "REAL"}
```

C.2.12 Тегирование

До введения конструкции AUTOMATIC TAGS спецификации АСН.1 часто содержали теги. Ниже описывается способ, с которым обычно применялось тегирование. С введением конструкции AUTOMATIC TAGS в новых спецификациях АСН.1 нет необходимости использовать обозначения тегов, хотя при модификациях старой нотации следует позаботиться о тегах.

C.2.12.1 Теги универсального класса используются только в настоящем стандарте. Например нотация [UNIVERSAL 30] предназначена исключительно для обеспечения точности в определении международно стандартизованных полезных типов. Иначе она не должна использоваться.

C.2.12.2 Часто встречающийся стиль использования тегов — присвоение тега прикладного класса ровно один раз во всей спецификации и использование его для идентификации типа, который широко используется во всей спецификации. Тег прикладного класса также часто используется в качестве тега типа в самом внешнем выборе CHOICE приложения, обеспечивая идентификацию отдельных сообщений через тег прикладного класса. Пример использования тега в последнем случае:

```
FileName : : = [APPLICATION 8] SEQUENCE  {
    directoryName                   VisibleString,
    directoryRelativeFileName       VisibleString }
```

C.2.12.3 Контекстно зависящее тегирование часто применяется алгоритмическим образом ко всем компонентам множества SET, последовательности SEQUENCE или выбора CHOICE. Однако с помощью возможности AUTOMATIC TAGS это делается намного проще.

Пример

```
CustomerRecord : : = SET {
    name                [0] VisibleString,
    maillingAddress     [1] VisibleString,
    accountNumber       [2] INTEGER,
    balanceDue          [3] INTEGER       }
CustomerAttribute : : = CHOICE {
    name                [0] VisibleString,
    maillingAddress     [1] VisibleString,
    accountNumber       [2] INTEGER,
    balanceDue          [3] INTEGER       }
```

C.2.12.4 Тегирование пользовательского класса обычно не используется в спецификациях стандартов (хотя это и не запрещается). Приложения, созданные производителями, обычно будут использовать теги прикладного и контекстно зависящего классов. Однако в редких случаях спецификация конкретного производителя будет расширять спецификацию стандарта, и в этих случаях использование тегов пользовательского класса может дать некоторые преимущества в части защиты спецификации производителя от изменений спецификации стандарта.

Пример

```
    AcmeBadgeNumber : : =  [PRIVATE 2] INTEGER
    badgeNumber AcmeBadgeNumber : : = 2345
```

C.2.12.5 Текстуальное использование IMPLICIT с каждым тегом можно найти только в старых спецификациях. Когда используется явное тегирование, правила BER создают менее компактное представление, чем при неявном тегировании. Правила PER создают одинаково компактное тегирование в обоих случаях. При использовании правил BER и явного тегирования в закодированных данных лучше видны нижележащие типы (INTEGER, REAL, BOOLEAN и прочие). В примерах настоящего руководства, когда возможно, используется неявное тегирование. Это может, в зависимости от правил кодирования, привести к компактному представлению, которое весьма желательно в некоторых приложениях. В других приложениях компактность может быть менее важна, чем, например, возможность осуществлять строгую проверку типов. В таком случае может использоваться явное тегирование.

Пример

```
CustomerRecord : : = SET {
    name                [0] IMPLICIT VisibleString,
    maillingAddress     [1] IMPLICIT VisibleString,
    accountNumber       [2] IMPLICIT INTEGER,
    balanceDue          [3] IMPLICIT INTEGER }
CustomerAttribute : : = CHOICE {
    name                [0] IMPLICIT VisibleString,
    maillingAddress     [1] IMPLICIT VisibleString,
    accountNumber       [2] IMPLICIT INTEGER,
    balanceDue          [3] IMPLICIT INTEGER }
```

C.2.12.6 Руководство по использованию тегов в новых спецификациях АСН.1, ссылающихся на настоящий стандарт, очень простое: НЕ ИСПОЛ ЗУЙТЕ ТЕГИ. Вставьте в заголовок модуля AUTOMATIC TAGS —

и забудьте о тегах. Если в последующей версии вам необходимо добавить новые компоненты к множеству SET, последовательности SEQUENCE или выбору CHOICE, добавьте их в конце.

С.2.13 Выбор

С.2.13.1 Выборочный тип CHOICE используется для моделирования переменных, выбираемых из совокупности переменных, число которых известно и невелико.

Пример

```
FileIdentifier : : = CHOICE {              - - идентификатор файла
    relativeName            VisibleString,
      - - имя файла (например, "MarchProgressReport")
    absoluteName            VisibleString,
      - - имя файла и содержащего его каталога
      - - (например, "<Williams> MarchProgressReport")
    serialNumber            INTEGER
      - - системный идентификатор файла - - }
file FileIdentifier : : = serialNumber : 106448503
```

С.2.13.2 Расширяемый выборочный тип CHOICE используется для моделирования переменных, выбираемых из совокупности переменных, разметка которой, вероятно, будет меняться от одной версии протокола к другой.

Пример

```
FileIdentifier : : = CHOICE {  - - Первая версия FileIdentifier
    relativeName      VisibleString,
    absoluteName      VisibleString,
    ..., ...
}
fileId1 FileIdentifier : : = relativeName : "MarchProgressReport.doc"
```

в предвидении:

```
FileIdentifier : : = CHOICE {       - - Вторая версия FileIdentifier
    relativeName      VisibleString,
    absoluteName      VisibleString,
    ...,
    serialNumber      INTEGER,  - - Расширяющее дополнение,
                                - - к второй версии
    ...
}
fileId1 FileIdentifier : : = relativeName : "MarchProgressReport.doc"
fileId2 FileIdentifier : : = serialNumber : 214
```

и позже:

```
FileIdentifier : : = CHOICE {- - Третья версия FileIdentifier
    relativeName      VisibleString,
    absoluteName      VisibleString,
    ...,
    serialNumber      INTEGER,  - - Расширяющее дополнение,
                                - - к второй версии
    [[      - - Расширяющее дополнение к третьей версии
      vendorSpecific          VendorExt,
      unidentified            NULL
    ]],
    ...
}
fileId1 FileIdentifier : : = relativeName : "MarchProgressReport.doc"
fileId2 FileIdentifier : : = serialNumber : 214
fileId3 FileIdentifier : : = unidentified : NULL
```

С.2.13.3 Расширяемый выборочный тип CHOICE из единственного типа используется, когда рассматривается возможность, что в будущем будут допустимы несколько типов.

Пример

```
Greeting : : = CHOISE {                    - - Первая версия Greeting
    postCard      VisibleString
    ...,
    ...
}
```

в предвидении:

```
Greeting : : = CHOISE {                    - - Вторая версия Greeting
        postCard        VisibleString,
        ...,
        [[    - - Расширяющее дополнение к второй версии
          audio         Audio,
          video         Video
        ...
}
```

С.2.13.4 Когда одно выборочное значение вложено в другое выборочное значение, требуется несколько двоеточий.

Пример

```
Greeting : : = [APPLICATION 12] CHOISE {
        postCard        VisibleString,
        recording       Voice }
Voice : : = CHOICE {
        english                         OCTET STRING,
        swahili                         OCTET STRING }
myGreeting Greeting : : = recording : english : '019838547EO'H
```

С.2.14 Селективный тип

С.2.14.1 Селективный тип используется для моделирования переменной, тип которой есть тип некоторой конкретной альтернативы в определенном ранее выборе CHOICE.

С.2.14.2 Рассмотрим определение:

```
FileAttribute : : = CHOICE {               - - атрибуты файла
    date-last-used          INTEGER,  - - дата последнего использования
    file-name               VisibleString }      - -  имя файла
```

Тогда возможно следующее определение

```
AttributeList : : = SEQUENCE {          - - список атрибутов
     first-attribute          date-last-used < FileAttribute,
     second-attribute         file-name < FileAttribute }
```

с возможной нотацией значения

```
listOfAttributes AttributeList : : = {
     first-attribute          27,
     second-attribute         "PROGRAM" }
```

С.2.15 Тип "поле класса объектов"

С.2.15.1 Тип "поле класса объектов" используется для идентификации типа, определенного с помощью класса информационных объектов (см. ГОСТ Р ИСО/МЭК 8824-2). Например поля класса информационных объектов ATTRIBUTE могут быть использованы в определении типа Attribute.

Пример

```
ATTRIBUTE : : = CLASS
{
   &AttributeType,
   &attributeId                 OBJECT IDENTIFIER UNIQUE
}
Attribute : : = SEQUENCE {
   attributeID ATTRIBUTE.&attributeId,    - - обычно ограничен
   attributeValue  ATTRIBUTE.&attributeType      - - обычно ограничено
}
```

Как ATTRIBUTE.&attributeId, так и ATTRIBUTE.&AttributeType являются типами полей класса объектов, определенными указанием класса информационных объектов ATTRIBUTE. Тип ATTRIBUTE.&attributeId фиксированный потому, что он явно определен в ATTRIBUTE как OBJECT IDENTIFIER. Однако тип ATTRIBUTE.&attributeType может передавать значения любого типа, определенного с использованием АСН.1, так как его тип не зафиксирован в определении ATTRIBUTE. Нотации, которые представляют данное свойство (передавать значение любого типа), называются "нотациями открытого типа", следовательно ATTRIBUTE.&AttributeType является открытым типом.

С.2.16 Встроенное-здп

С.2.16.1 Тип "встроенное-здп" используется для моделирования переменных, тип которых не задан или задан где-либо в другом месте, без ограничения на нотацию, используемую для спецификации типа.

Пример

```
FileContents : : = EMBEDDED PDV          - - содержимое файла
DocumentList : : = SEQUENCE OF EMBEDDED PDV - - список документов
```

С.2.17 Внешний тип

Внешний тип похож на встроенное-здп, но имеет меньше идентификационных опций. В новых спецификациях предпочтительнее использовать встроенное-здп из-за его большей гибкости и того обстоятельства, что некоторые правила кодирования представляют его значения более эффективно.

С.2.18 Экземпляр-из

С.2.18.1 Экземпляр-из используется для спецификации типа, содержащего поле идентификатора объекта и открытый тип, значение которого есть тип, определенный идентификатором объекта. Тип "экземпляр-из" ограничен тем, что может переносить значение из класса TYPE-IDENTIFIER (см. ГОСТ Р ИСО/МЭК 8824-2, приложения А и С).

Пример

```
ACCESS-CONTROL-CLASS : : = TYPE-IDENTIFIER
Get-Invoke : : = SEQUENCE {
    objectClass             objectClass,
    objectInstance          ObjectInstance,
    accessControl           INSTANCE OF ACCESS-CONTROL-CLASS,      - - обычно ограничен
    attributeID             ATTRIBUTE.&attributeId
}
```

Конструкция GET-Invoke эквивалентна следующей:

```
Get-Invoke : : = SEQUENCE {
    objectClass             objectClass,
    objectInstance          ObjectInstance,
    accessControl           [UNIVERSAL 8] IMPLICIT SEQUENCE  {
        type-id             ACCESS-CONTROL-CLASS.&id,          - - обычно ограничен
        value               [0] ACCESS-CONTROL-CLASS.&Type     - - обычно ограничен
    },
    attributeID             ATTRIBUTE.&attributeId
}
```

Действительное предназначение типа "экземпляр-из" не видно до тех пор, пока он не ограничивается с использованием множества информационных объектов, но такие примеры выходят за рамки настоящего стандарта. Определение множества информационных объектов см. в ИСО/МЭК 8824-3, а в приложении А к нему — использование множества информационных объектов для ограничения типа "экземпляр-из". Кодирование INSTANCE OF ACCESS-CONTROL-CLASS то же самое, что и для значения EXTERNAL, которое содержит только идентификатор объекта и значение данных.

С.3 Идентификация абстрактных синтаксисов

С. 3.1 Использование услуг уровня представления (ГОСТ 34.971) требует спецификации значений, называемых значениями данных (уровня) представления, и объединений этих значений данных представления в множества, называемые абстрактными синтаксисами. Каждому из этих множеств дано имя абстрактного синтаксиса типа идентификатор объекта ACH.1.

С.3.2 ACH.1 может использоваться как общий инструмент для спецификации значений данных представления и их объединения в поименованные абстрактные синтаксисы.

С.3.3 В простейшем случае такого использования имеется единственный тип ACH.1, такой, что каждое значение данных представления в поименованном абстрактном синтаксисе является значением этого типа ACH.1. Этот тип обычно является выборочным типом, а каждое значение данных представления будет альтернативой из этого выборочного типа. В данном случае рекомендуется, чтобы используемая нотация модуля ACH.1 содержала этот выборочный тип в качестве первого определяемого типа с последующими определениями (не универсальных) типов, которые прямо или косвенно указываются в этом выборочном типе.

Примечание — Сказанное не подразумевает исключение ссылок на типы, определенные в других модулях.

С.3.4 Рекомендуется, чтобы присваивание идентификатора и описателя объекта абстрактному синтаксису осуществлялось с использованием полезного класса информационных объектов ABSTRACT-SYNTAX, определенного в ГОСТ Р ИСО/МЭК 8824-2. Так же рекомендуется, чтобы все использования ABSTRACT-SYNTAX были сгруппированы в одном "корневом" модуле, идентифицирующем все абстрактные синтаксисы, используемые в прикладном стандарте.

С.3.5 Ниже приводится пример текста, который может встретиться в прикладном стандарте.

Пример

```
ISOxxxx-yyyy {iso standard xxxx ans1-modules (...) yyyy-pdu (...)} DEFINITIONS : : =
BEGIN
    EXPORTS YYYY — PDU;
    YYYY — PDU : : = CHOICE {
        connect-pdu……,
        data-pdu CHOICE  {
```

```
                    ......,
                    .....
            },
            .....
        }
        .....
    END
    ISOxxxx-yyyy-Abstract-Syntax-Module {iso standard xxxx ans1-modules (...)} DEFINITIONS : : =
    BEGIN
        IMPORTS YYYY-PDU FROM ISOxxxx-yyyy {iso standard xxxx ans1-modules (...) yyyy-pdu (...)};
- - В настоящем стандарте определен следующий абстрактный синтаксис:
    YYYY-Abstract-Syntax ABSTRACT-SYNTAX : : =
        {YYYY-PDU IDENTIFIED BY yyyy-abstract-syntax-object-id}
    yyyy-abstract-syntax-object-id OBJECT IDENTIFIER : : = {
        iso standard yyyy (xxxx) abstract-syntax (. . .)}
- - Соответствующим описателем объекта является
    yyyy-abstract-syntax-descriptor ObjectDescriptor : : = "....."
- - Значения идентификатора и описателя объекта АСН.1:
                        - - идентификатор объекта правила кодирования,
                        - - описатель объекта правила кодирования,
- - присвоенные правилам кодирования в ИСО/МЭК 8825-1 и ИСО/МЭК 8825-2 могут использоваться как
- - идентификатор синтаксиса передачи вместе с настоящим абстрактным синтаксисом,
- - ISOxxxx-yyyy-Abstract-Syntax
    END
```

С.3.6 Для того чтобы гарантировать взаимодействие, стандарт может дополнительно потребовать обязательной поддержки синтаксиса передачи, полученного применением правил кодирования, указанных в модуле абстрактного синтаксиса.

С.4 Подтипы

С.4.1 Подтипы используются для ограничения значений существующего типа, допустимых в конкретной ситуации.

Примеры

```
AtomicNumber : : = INTEGER (1..104) - - атомное число
TouchToneString : : = IA5String - - строка набора номера
        (FROM ("0123456789" | "H" | "#")) (SIZE (1. . 63))
ParameterList : : = SET SIZE (1. . 63) OF Parameter - - список параметров
SmallPrime : : = INTEGER (2 | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29) - - малое простое число
```

С.4.2 Расширяемое ограничение подтипа используется для моделирования типа INTEGER, множество допустимых значений которого мало и четко определено, но может быть расширено.

Пример

```
SmallPrime : : = INTEGER (2 | 3, . . .)  - - Первая версия SmallPrime
```

в предвидении:

```
SmallPrime : : = INTEGER (2 | 3, . . . , 5 | 7 | 11)  - - Вторая версия SmallPrime
```

и позже:

```
SmallPrime : : = INTEGER (2 | 3, . . . , 5 | 7 | 11 | 13 | 17 | 19)  - - Третья версия SmallPrime
```

Примечание — Для некоторых типов одни правила кодирования (например PER) обеспечивают высокооптимизированное кодирование значений для ограничения подтипа корня расширения (т. е. значений, стоящих до ". . . ") и менее оптимизированное кодирование значений для ограничения подтипа расширяющих дополнений (т. е. значений, стоящих после ". . ."), тогда как при других правилах кодирования (например BER) ограничения подтипов не влияют на кодирование.

С.4.3 Когда два или более связанных типа имеют существенную общность, следует рассмотреть возможность явного определения этой общности как порождающего типа и введения подтипов для отдельных типов. Такой подход делает очевидными взаимосвязь и общность и продолжает их поддерживать для типов, имеющих отношение к делу. Тем самым обеспечивается возможность использования при реализации общего подхода к обработке значений этих типов.

Пример

```
Envelope : : = SET {
                typeA TypeA,
                typeB TypeB OPTIONAL,
                typeC TypeC OPTIONAL }
    - - общий порождающий тип
ABEnvelope : : = Envelope (WITH COMPONENTS
```

```
                {...,
                    typeB PRESENT, typeC ABSENT })
    - - typeB всегда должен присутствовать, typeC — отсутствовать
ACEnvelope : : = Envelope (WITH COMPONENTS
                {...,
                    typeB ABSENT, typeC PRESENT })
    - - typeC всегда должен присутствовать, typeB — отсутствовать
```

Два последних определения альтернативно могут быть записаны как:

```
ABEnvelope : : = Envelope (WITH COMPONENTS {typeA, type B})
ACEnvelope : : = Envelope (WITH COMPONENTS {typeA, type C})
```

Выбор между этими альтернативами мог бы быть сделан на основе таких факторов, как количество компонентов в порождающем типе, количество среди них факультативных, степень различия отдельных типов и, вероятно, стратегии развития.

С.4.4 Подтипы используются для частично определенных значений, например для протокольных блоков данных (ПБД), которые должны быть протестированы в тесте соответствия, когда тест направлен только на некоторые из компонентов ПБД.

П р и м е р

Пусть определен ПБД

```
PDU : : = SET
    {alpha       INTEGER,
    beta         IA5String OPTIONAL,
    gamma        SEQUENCE OF Parameter,
    delta        BOOLEAN }
```

Тогда при составлении теста, требующего, чтобы булевское значение было ложным, а целое — отрицательным, записывается:

```
TestPDU : : = PDU (WITH COMPONENTS
    {...,
    delta (FALSE),
    alpha (MIN. . < 0)})
```

Далее, если строка IA5String (beta) должна присутствовать и быть длиной 5 или 12 символов, то записывается:

```
FurtherTestPDU : : = TestPDU  (WITH COMPONENTS {. . . , beta (SIZE (5 | 12)) PRESENT})
```

С.4.5 Если тип данных общего назначения был определен как SEQUENCE OF, то подтипы используются для определения ограниченных подтипов общего типа.

П р и м е р

```
Text-block : : = SEQUENCE OF VisibleString
Address : : = Text-block (SIZE (1. . 6)) (WITH COMPONENT  (SIZE  (1 . . 32)))
```

С.4.6 Если тип данных общего назначения был определен как CHOICE, то подтипы используются для определения ограниченных подтипов общего типа.

П р и м е р

```
Z : : = CHOICE {
        a       A,
        b       B,
        c       C,
        d       D,
        e       E
}
V : : = Z  (WITH COMPONENTS {. . . , a ABSENT, b ABSENT})
                - - 'a' и 'b' должны отсутствовать;
                - - 'c', 'd' или 'e' могут присутствовать в значении
W : : = Z (WITH COMPONENTS {. . . , a PRESENT})
                - - может присутствовать только 'a' (см. 48.8.9.2)
X : : = Z (WITH COMPONENTS { a PRESENT})
                - - может присутствовать только 'a' (см. 48.8.9.2)
Y : : = Z (WITH COMPONENTS {a ABSENT, b, c})
                - - 'a', 'd' и 'e' должны отсутствовать;
                - - 'c' или 'b' могут присутствовать в значении
```

П р и м е ч а н и е — W и X семантически идентичны.

С.4.7 Подтипы используются для образования новых подтипов из существующих.

П р и м е р

```
Months : : = ENUMERATED {
                january         (1),
```

```
                february        (2),
                march           (3),
                april           (4),
                may             (5),
                june            (6),
                july            (7),
                august          (8),
                september       (9),
                october         (10),
                november        (11),
                december        (12) }
First-quarter : : = Months (
                        january         |
                        february        |
                        march )
Second-quarter : : = Months (
                        april           |
                        may             |
                        june )
Third-quarter : : = Months (
                        july            |
                        august          |
                        september )
Fourth-quarter : : = Months (
                        october         |
                        november        |
                        december )
First-half : : = Months (First-quarter | Second-quarter)
Second-half : : = Months (Third-quarter | Fourth-quarter)
```

ПРИЛОЖЕНИЕ D
(справочное)

Руководство по использованию символьных строк АСН.1

D.1 Поддержка символьных строк в АСН.1

D.1.1 Имеются следующие четыре группы символьных строк, поддерживаемых в АСН.1:

а) типы символьных строк, основанные на Международном регистре ИСО наборов кодированных символов, которые должны использоваться с Escape-последовательностями (то есть на структуре ИСО 646), и на связанном с ним Международном регистре кодовых символьных наборов, которые обеспечиваются типами VisibleString, IA5String, TeletexString, VideotexString, GraphicString и GeneralString;

б) типы символьных строк, основанные на ИСО/МЭК 10646-1, которые обеспечиваются типами UniversalString, UTF8String и BMPString с подмножествами, определенными в ИСО/МЭК 10646-1, или использованием поименованных символов.

Примечания

1 Использование типа UniversalString без ограничений приводит к нарушению требований соответствия для информационного обмена, специфицированных в ИСО/МЭК 10646-1, так как при этом заданы не принимаемые подмножества.

2 Несмотря на сказанное выше, использование этого типа с простым ограничением подтипа, которое использует параметр абстрактного синтаксиса (для ограничения определенным подтипом UniversalString), может обеспечить мощный и гибкий механизм обработки символов, доверяя профилям определение значения параметра для удовлетворения конкретных потребностей сообщества пользователей. Однако в общем случае в стандартах предпочтительнее использовать тип CHARACTER STRING (см. ниже);

в) типы символьных строк, обеспечивающие простые небольшие совокупности символов, определенные в настоящем стандарте и предназначенные для специализированного использования; к ним относятся типы NumericString и PrintableString;

г) тип CHARACTER STRING с согласованным (или объявленным) символьным набором, который будет использоваться; это позволяет реализации использовать любую совокупность символов и кодирований, которым присвоены OBJECT IDENTIFIER, включая совокупности из Международного регистра ИСО наборов кодированных символов, которые должны использоваться с Escape-последовательностями, ИСО/МЭК 7350, ИСО/МЭК 10646-1 и пользовательские совокупности символов и кодирования (профили могут накладывать требования или ограничения на символьные наборы — символьные абстрактные синтаксисы, которые могут использоваться).

D.2 Типы UniversalString, UTF8String и BMPString

D.2.1 Типы UniversalString и UTF8String переносят любые символы из ИСО/МЭК 10646-1. Наборы символов в ИСО/МЭК 10646-1, в общем случае, слишком велики для разумных требований соответствия и обычно должны быть выделены их поднаборы в виде комбинаций стандартных совокупностей символов из приложения А ИСО/МЭК 10646-1.

D.2.2 Тип BMPString переносит любые символы из основной многоязычной плоскости ИСО/МЭК 10646-1 (первые 62К — 2 символов). Основная многоязычная плоскость обычно разбивается на поднаборы в виде комбинаций стандартных совокупностей символов из приложения А ИСО/МЭК 10646-1.

D.2.3 Для совокупностей, определенных в ИСО/МЭК 10646-1, приложение А, имеются ссылки на тип, приведенные во встроенном модуле АСН.1 "ASN1-CHARACTER-MODULE" (см. раздел 37). Метод "ограничения подтипа" позволяет определять новые подтипы UniversalString, которые являются комбинациями существующих подтипов.

D.2.4 Примерами ссылок на тип, определенных в ASN1-CHARACTER-MODULE, и соответствующих им имен совокупностей в ИСО/МЭК 10646-1 являются:

BasicLatin	BASIC LATIN
Latin-1Supplement	LATIN-1 SUPPLEMENT
LatinExtended-a	LATIN EXTENDED-A
LatinExtended-b	LATIN EXTENDED-B
IpaExtensions	IPA EXTENSIONS
SpacingModifierLetters	SPACING MODIFIER LETTERS
CombiningDiacriticalMarks	COMBINING DIACRITICAL MARKS

D.2.5 В ИСО/МЭК 10646-1 определены три "уровня реализации" и требуется, чтобы все пользователи ИСО/МЭК 10646-1 устанавливали уровень реализации.

Уровень реализации относится к степени, в которой поддерживаются комбинированные символы и, следовательно, в терминах АСН.1, определяется подмножество UniversalString и BMPString, ограничивающее типы символьных строк.

Для уровня реализации 1 комбинированные символы не допустимы, и обычно имеется взаимно однозначное соответствие между абстрактными символами (ссылками на ячейки) в символьной строке АСН.1 и печатными символами в физическом представлении строки.

Для уровня реализации 2 допускается использование некоторых комбинированных символов (перечисленных в ИСО/МЭК 10646-1, приложение В), но использование других таких символов запрещено.

Для уровня реализации 3 нет ограничений на использование комбинированных символов.

D.2.6 Типы BMPString и UniversalString могут быть ограничены с помощью нотации подтипа так, чтобы исключить все управляющие функции:

```
VanillaBMPString : : = BMPString (FROM (ALL EXCEPT
                        ({0, 0, 0, 0} . . {0, 0, 0, 31} ! {0, 0, 0, 128} . . {0, 0, 0, 159})))
```

или, эквивалентно,

```
C0 : : = BMPString (FROM ({0, 0, 0, 0} . . {0, 0, 0, 31}))      - - Функции C0
C1 : : =  BMPString (FROM ({0, 0, 0, 128} . . {0, 0, 0, 159})) - - Функции C1
VanillaBMPString : : = BMPString (FROM (ALL EXCEPT  (C0 | C1)))
```

D.3 О требованиях соответствия ИСО/МЭК 10646-1

Использование UniversalString, BMPString или UTF8String (или их подтипов) в определении типа АСН.1 требует обращения к требованиям соответствия ИСО/МЭК 10646-1.

Согласно этим требованиям соответствия разработчики стандарта (скажем, X), использующие такие типы АСН.1, должны обеспечивать (в заявке о соответствии реализации протоколу) утверждение о принимаемом подмножестве ИСО/МЭК 10646-1 для реализаций их стандарта X или об уровне поддержки комбинированных символов этими реализациями.

Использование подтипа UniversalString, UTF8String или BMPString в спецификации требует, чтобы реализации поддерживали все символы ИСО/МЭК 10646-1, включенные в этот подтип АСН.1, и, следовательно, чтобы (по крайней мере) эти символы присутствовали в принимаемом подмножестве для реализации. Так же требуется, чтобы установленный уровень поддерживался всеми такими подтипами АСН.1.

Примечание — Спецификация АСН.1 (при отсутствии параметров абстрактного синтаксиса и спецификации исключений) определяет как максимальный набор символов, которые могут быть переданы, так и минимальный набор символов, которые должны обрабатываться получателем. Принимаемый набор ИСО/МЭК 10646-1 требует, чтобы символы не из этого набора не передавались, а все символы этого набора поддерживались получателем. Следовательно, необходимо, чтобы принимаемый набор был бы в точности набором всех символов, допускаемых спецификацией АСН.1. Случай, когда имеется параметр абстрактного синтаксиса, рассматривается ниже.

D.4 Рекомендации пользователям АСН.1 по соответствию ИСО/МЭК 10646-1

Пользователи АСН.1 должны четко устанавливать набор символов ИСО/МЭК 10646-1, который образует принимаемый поднабор реализации (и требуемый уровень реализации) для того, чтобы выполнялись требования их стандарта.

Это можно удобно сделать путем определения подтипа UniversalString, UTF8String или BMPString, который содержит все необходимые для стандарта символы, и ограничения его "уровнем 1" или "уровнем 2". Подходящим именем для этого типа может быть "ISO-10646-String".

Пример

```
ISO-10646-String : : =  BMPString
                    (FROM (Level2 INTERSECTION (BasicLatin UNION HebrewExtended UNION Hiragana)))
- - Это тип, который определяет минимальный набор символов в принимаемом поднаборе для
- - реализации настоящего стандарта.
- - Требуемый уровень реализации — не ниже 2.
```

Тогда ЗСРП будет содержать простое утверждение, что принимаемый поднабор ИСО/МЭК 10646-1 есть ограниченный поднабор (и уровень), определенный типом "ISO-10646-String", а "ISO-10646-String" (возможно, его подтипы) будет использоваться в стандарте всюду, где требуется включить строки ИСО/МЭК 10646-1.

Пример ЗСРП

Принимаемый поднабор ИСО/МЭК 10646-1 есть ограниченный поднабор, состоящий из всех символов в типе АСН.1 "ISO-10646-String", определенном в модуле <имя модуля>, с уровнем реализации 2.

Пример использования в протоколе:

```
Message : : = SEQUENCE {
first-field ISO-10646-String     - - допустимы все символы из принимаемого поднабора
second-field ISO-10646-String (FROM (latinSmallLetterA . . latinSmallLetterZ)),
                                 - - допустимы только строчные латинские буквы
   third-field ISO-10646-String (FROM (digitZero . . digitNine))
                                 - - допустимы только цифры
}
```

D.5 Принимаемые поднаборы как параметры абстрактного синтаксиса

Стандарт ИСО/МЭК 10646-1 требует, чтобы принимаемый поднабор и уровень реализации были определены явно. Когда пользователь АСН.1 не хочет ограничивать диапазон символов ИСО/МЭК 10646-1 в некоторой части разрабатываемого стандарта, то это можно сделать, определяя (например) "ISO-10646-String" как подтип UniversalString, UTF8String или BMPString с ограничением подтипа, состоящим из (или содержащим) "ImplementorsSubset", который остается параметром абстрактного синтаксиса.

Пользователи АСН.1 должны учитывать, что в этом случае соответствующий отправитель может передать соответствующему получателю символы, которые не могут быть обработаны получателем потому, что они выпадают из (зависящего от реализации) принимаемого поднабора или уровня получателя, и рекомендуется, чтобы в этом случае в определение "ISO-10646-String" включалась спецификация обработки исключений.

Пример

```
ISO-10646-String {UniversalString : ImplementorsSubset, ImpementationLevel} : : =
      UniversalString (FROM ((ImplementorsSubset UNION BasicLatin)
                  INTERSECTION ImplementationLevel) characterSetPrblem)
   - - Принимаемый поднабор ИСО/МЭК 10646-1 должен содержать "BasicLatin", но может содержать
   - - любые дополнительные символы, определенные в  "ImplementorsSubset", который является парамет-
   - - ром абстрактного синтаксиса. "ImpementationLevel" является параметром абстрактного синтаксиса,
   - - определяющим уровень реализации. Соответствующий получатель должен быть готов получить
   - - символы вне этого принимаемого поднабора и уровня реализации. Обработка исключения в этом
   - - случае определена в разделе <номер раздела> для вызова "characterSetPrblem". Она может никогда не
   - - вызываться соответствующим получателем, если символы, фактически используемые в сеансе
   - - взаимодействия, ограничены совокупностью "BasicLatin".
My-Level2-String : : = ISO-10646-String {{HebrewExtended UNION Hiragana}, Level2}
```

D.6 Тип CHARACTER STRING

D.6.1 Тип CHARACTER STRING предоставляет полную гибкость в выборе символьного набора и метода кодирования. Когда единственное соединение обеспечивает сквозную передачу данных (без прикладной ретрансляции), тогда должно использоваться согласование символьных наборов, а кодирование может быть выполнено как часть определения контекстов представления для символьных абстрактных синтаксисов.

D.6.2 Важно понимать, что символьный абстрактный синтаксис является обычным абстрактным синтаксисом с некоторыми ограничениями на возможные значения (они все являются символьными строками и всеми символьными строками, образованными из некоторой совокупности символов). Таким образом, регистрация таких синтаксисов и согласование контекста представления осуществляется обычным путем.

D.6.3 Кодирование типа CHARACTER STRING также допускает объявление используемых абстрактного синтаксиса и синтаксиса передачи без согласования в том окружении, в котором это применимо.

Примечания

1 Проектировщики приложений могут запретить использование согласования представления, сделать его обязательным или оставить на усмотрение отправителя.

2 Когда используется объявление, а не согласование, проектировщик приложения должен рассмотреть, как отправитель может определить, что символьный абстрактный синтаксис (и синтаксис передачи) может быть приемлемым для получателя (например используя услуги справочника или в результате профилирования), а так же рассмотреть действия получателя, если получено значение CHARACTER STRING из символьного абстрактного синтаксиса, который не поддерживается.

D.6.4 Если используется согласование, то проектировщик прикладного уровня может управлять таким согласованием, специфицируя, когда должны быть установлены такие контексты представления, и параметр пользовательских данных примитива P-ALTER-CONTEXT, или может просто принять, что некоторый профиль должен определить, какой символьный абстрактный синтаксис должен использоваться, устанавливая для него контекст представления в момент передачи примитива P-CONNECT.

D.6.5 Возможности управления контекстом услуг уровня представления позволяют инициатору (в P-CONNECT или в установленном соединении, используя P-ALTER-CONTEXT) предложить список новых или исключаемых из использования абстрактных синтаксисов (который может содержать символьные абстрактные синтаксисы), а получателю — выбрать из этого списка.

D.6.6 Инициатор может выразить свои предпочтения порядком абстрактных синтаксисов в списке или через параметр пользовательских данных, который предоставлен проектировщиком приложения для того, чтобы пояснить цель предлагаемого использования нового абстрактного синтаксиса. Например он может указывать, что все (символьные) абстрактные синтаксисы предлагаются для использования с одной единственной целью или что подразумевается выбор единственного (символьного) абстрактного синтаксиса для использования в различных целях.

D.6.7 Символьные абстрактные синтаксисы (и соответствующие символьные синтаксисы передачи) определяются в ряде стандартов (или рекомендаций МСЭ-Т), а дополнительные символьные абстрактные синтаксисы (и/или символьные синтаксисы передачи) могут быть определены любой организацией, выделяющей идентификаторы объектов.

D.6.8 В ИСО/МЭК 10646-1 имеется символьный абстрактный синтаксис, определенный (и ему присвоен идентификатор объекта) для всей совокупности символов, каждой определенной совокупности символов, поднаборов (BASIC LATIN, BASIC SYMBOLS, и т. д.) и любой возможной комбинации определенных совокупностей символов. В ИСО/МЭК 10646-1 имеются так же два символьных синтаксиса передачи, определенные для идентификации различных опций (в частности, 16- и 32-битовых).

ПРИЛОЖЕНИЕ Е
(справочное)

Замененные характеристики

Ряд характеристик, включенных в предшествующую редакцию настоящего стандарта (а именно в ГОСТ Р ИСО/МЭК 8824), были заменены и теперь не являются частью АСН.1. Однако они могут встретиться в некоторых существующих модулях АСН.1. В настоящем приложении описаны эти характеристики и то, как их возможности могут быть реализованы с использованием заменивших их характеристик.

Е.1 Использование идентификаторов является обязательным

Нотациями для NamedType и NamedValue первоначально были:

NamedType : : = identifier Type | Type | SelectionType

NamedValue : : = identifier Value | Value

но теперь они изменены на:

NamedType : : = identifier Type

NamedValue : : = identifier Value

так как прежние приводили к двусмысленной грамматике.

Идентификаторы могут быть добавлены к поименованным типам NamedType в старых спецификациях АСН.1 без влияния на кодирование типа (хотя изменения АСН.1 будут необходимы для всех использований соответствующих нотаций значений). Такие изменения могут быть даны либо в сообщении об ошибках, либо как часть новой версии модифицируемого стандарта.

Е.2 Выборочное значение

Нотацией значения для выборочного типа первоначально была:

ChoiceValue : : = NamedValue

NamedValue : : = identifier Value | Value

но теперь она изменена на

ChoiceValue : : = identifier ":" Value

так как прежняя приводила к двусмысленной грамматике.

Е.3 Произвольный тип

Произвольный тип был определен в ранних версиях настоящего стандарта.

Обычно произвольный тип использовался для того, чтобы оставить "дыру" в спецификации, которая должна быть заполнена некоторой другой спецификацией. Нотацией была "AnyType", которая допускалась в качестве альтернативы для "Type" и определялась как:

AnyType : : = ANY | ANY DEFINED BY identifier

Нотация значения для произвольного типа была

AnyValue : : = Type Value

хотя позднее она была изменена на

AnyValue : : = Type : Value

так как прежняя приводила к трудностям при машинной обработке АСН.1.

Настоятельно рекомендовалось использовать вторую альтернативу нотации. В этой альтернативе (единственной возможной, когда произвольный тип был одним из типов компонентов множества или последовательности) некоторый другой компонент множества или последовательности (с указанным идентификатором "identifier") должен был указывать, своим целочисленным значением или значением "идентификатор объекта" (или выбором из них), фактический тип, управляющий произвольным компонентом. Отображение таких значений в конкретные типы АСН.1 могло бы выглядеть как некоторая "таблица", образующая часть абстрактного синтаксиса. При отсутствии "DEFINED BY identifier" (первая альтернатива нотации) в нотации не было бы указания, как может быть определен тип поля. Это часто приводило к спецификациям, в которых "дыра" продолжала существовать даже на стадии, когда предполагалась реализация.

Теперь произвольный тип заменен возможностью спецификации классов информационных объектов и последующим указанием полей классов информационных объектов в определениях типов (см. ГОСТ Р ИСО/МЭК 8824-2). Так как поля могут быть определены таким образом, чтобы допускался произвольный тип АСН.1, то обеспечивается основная возможность — оставлять "дыры" в спецификации. Однако новая характеристика допускает спецификацию "табличного ограничения", в которой конкретное "множество информационных объектов" (соответствующего класса информационных объектов) явно указывается как ограничение типа. Тем самым обеспечивается то, что раньше обеспечивалось конструкцией "ANY DEFINED BY identifier".

Кроме того, предоставляется некоторое предопределенное использование новых возможностей (см. ГОСТ Р ИСО/МЭК 8824-2), которое соответствует различным, обычно встречающимся образцам использования произвольного типа. Например последовательность, содержащая идентификатор объекта и произвольный тип, ранее часто использовавшаяся для передачи произвольного значения вместе с идентификацией его типа, может быть описана как

INSTANCE OF MUMBLE

где MUMBLE определяется как класс информационных объектов (а не как тип АСН.1):

MUMBLE: : = TYPE-IDENTIFIER

Эта нотация приводит к тому, что конструкция "INSTANCE OF MUMBLE" должна быть заменена идентификатором объекта для объекта класса MUMBLE вместе с типом, указанным идентификатором объекта. Пример см. в С.2.18.

Конкретные пары идентификаторов объектов и типов определяются как информационные объекты класса MUMBLE, а если требуется, то могут быть определены их конкретные множества и использованы для ограничения конструкции INSTANCE OF таким образом, что могут встречаться только объекты этого множества.

Макровозможности часто использовались как полуформальный способ определения таблиц информационных объектов для управления ассоциированным использованием произвольного типа и заменены новыми возможностями.

Е.4 Макровозможности

Макровозможности позволяли пользователю АСН.1 расширять нотацию путем определения макросов.

Основным использованием макровозможностей было определение нотации для спецификации информационных объектов. Теперь такая возможность включена непосредственно в АСН.1 (см. ГОСТ Р ИСО/МЭК 8824-2) без необходимости совершенно общей (и, соответственно, опасной) определяемой пользователем нотации.

Кроме того, единственным другим использованием нотации является, видимо, определение выражений, которые должны применяться с некоторыми параметрами для того, чтобы быть полностью определенными типами АСН.1. Теперь это обеспечивается более общими возможностями параметризации (см. ИСО/МЭК 8824-4).

ПРИЛОЖЕНИЕ F
(обязательное)

Правила совместимости типов и значений

Приложение должно использоваться главным образом разработчиками с целью гарантировать, что они идентично интерпретируют язык. Оно приведено, чтобы четко специфицировать, что допустимо в АСН.1, а что — нет, и позволить специфицировать точное значение, которое идентифицирует какое-либо имя ссылки на значение, и точное множество значений, которое идентифицирует какое-либо имя ссылки на множество типов или значений. Приложение не устанавливает определения допустимых преобразований нотаций АСН.1 для каких-либо целей, отличных от указанных выше.

F.1 Необходимость понятия "отображение значений" (введение)

F.1.1 Рассмотрим следующие определения АСН.1:

```
A : : = INTEGER
B : : = [1] INTEGER
C : : = [2] INTEGER (0. . 6, . . . )
D : : = [2] INTEGER (0. . 6, . . . , 7)
E : : = INTEGER (7. . 20)
F : : = INTEGER {red (0), white (1), blue (2), green (3), purple (4)}
a A : : = 3
b B : : = 4
c C : : = 5
d D : : = 6
e E : : = 7
f F : : = green
```

F.1.2 Ясно, что ссылки на значения a, b, c, d, e, f могут быть использованы в нотациях значений, управляемых A, B, C, D, E, F соответственно. Например:

```
W : : = SEQUENCE {w1      A      DEFAULT a}
x A : : = a
Y : : = A (1 . . a)
```

где все записи допустимы при определениях, данных в F.1.1. Однако если заменить A на B, C, D, E или F, то будут ли получившиеся предложения допустимыми? Аналогично, если ссылку на значение a заменить ссылкой b, c, d, e или f, то будут ли получившиеся предложения допустимыми?

F.1.3 Более сложный вопрос возник бы в случае замены ссылки на тип явным текстом правой части присваивания. Например:

```
f  INTEGER {red (0), white (1), blue (2), green (3), purple (4)} : : = green
W : : = SEQUENCE {
          w1 INTEGER {red (0), white (1), blue (2), green (3), purple (4)}
             DEFAULT f }
x  INTEGER {red (0), white (1), blue (2), green (3), purple (4)} : : = f
Y : : =  INTEGER {red (0), white (1), blue (2), green (3), purple (4)} (1 . . f)
```

Допустимо ли это в АСН.1?

F.1.4 Некоторые из приведенных выше примеров таковы, что даже если и являются допустимыми (а большинство из них допустимы — см. ниже), то пользователям не следовало бы писать аналогичные тексты, как как они мало понятны и вводят в заблуждение. Однако часто встречается использование ссылки на значение некоторого типа (не обязательно INTEGER), как на значение по умолчанию для этого типа с применением тегирования или образования подтипа в управляющем типе. Понятие "отображение значений" вводится для того, чтобы обеспечить ясный и точный смысл определения, какие из приведенных выше конструкций допустимы.

F.1.5 Рассмотрим определения

```
C : : = [2] INTEGER (0. . 6, . . . )
E : : = INTEGER (7. . 20)
F : : = INTEGER {red (0), white (1), blue (2), green (3), purple (4)}
```

В каждом случае создается новый тип. Для F можно четко идентифицировать соответствие 1—1 между значениями в этом типе и значениями в универсальном типе INTEGER. Для C и E можно четко идентифицировать соответствие 1—1 между значениями в этих типах и подмножествами значений в универсальном типе INTEGER. Это взаимоотношение называется отображением значений между значениями в двух типах. Более того, так как значения в C, E и F имеют отображения 1—1 в значения INTEGER, можно использовать эти отображения для обеспечения отображений между значениями самих C, E и F. Для C и F это показано на рисунке F.1.

F.1.6 Пусть имеется ссылка на значение, такая как

c C : : = 5

на значение в C, которая должна в некотором контексте идентифицировать значение в F. Тогда, при условии, что существует отображения значений между этим значением в C и (единственным) значением в F, можно определить C как допустимую ссылку на значение в F. Это показано на рисунке F.2, где ссылка на значение C используется для идентификации значения в F и может использоваться вместо непосредственной ссылки f1, которую, в противном случае, нужно было бы определить:

f1 F : : = 5

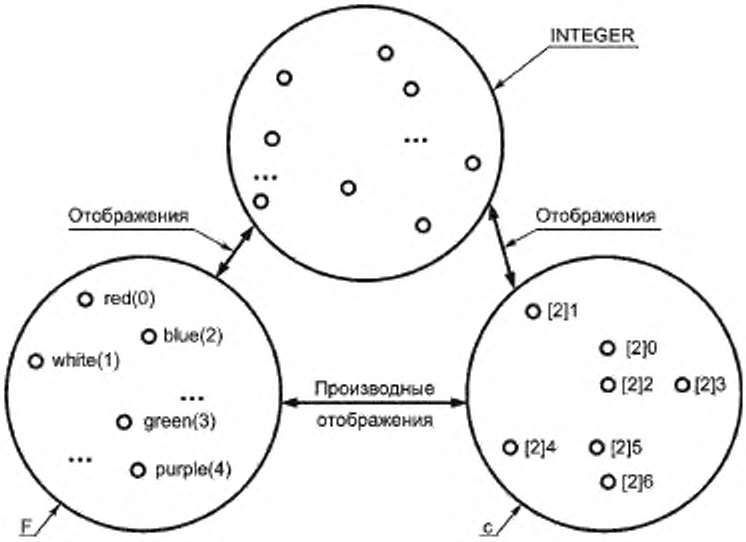

Рисунок F.1

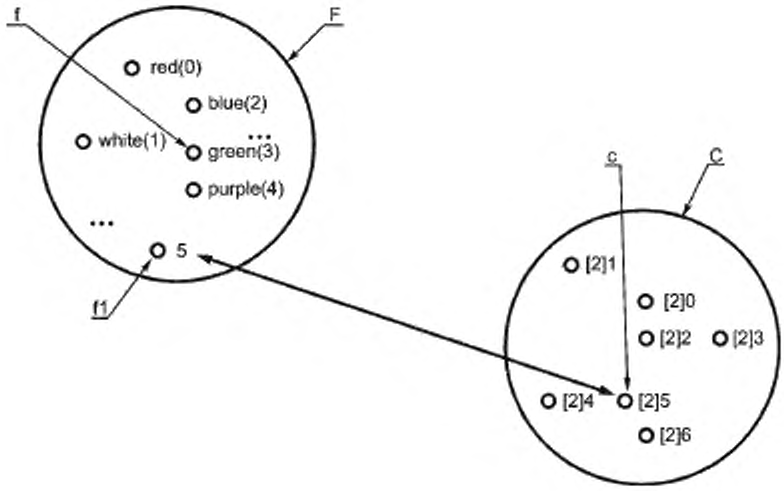

Рисунок F.2

F.1.7 Следует учитывать, что в некоторых случаях в одном типе имеются значения (например 7—20 в A, см. F.1.1), которые имеют отображения в значения в другом типе (например 7—20 в E, см. F.1.1), а другие значения (например равные или больше 21 в A, см. F.1.1) таких отображений не имеют. Ссылка на такое значение в A не даст допустимой ссылки на значение в E. (В данном примере весь тип E имеет отображение значений в подмножество A. В общем случае, в обоих типах могут быть подмножества значений, имеющих отображения, и подмножества значений без отображений).

F.1.8 В стандартах АСН.1 используется текст на естественном языке для спецификации допустимости в приведенных выше и аналогичных случаях. В F.6 даны точные требования для допустимости, к которым следует обращаться при затруднениях со сложными конструкциями.

Примечание — То обстоятельство, что определено отображение значений между двумя конструкциями "Type", позволяет использовать ссылки на значения, установленные с использованием одной конструкции "Type", для идентификации значений в другой конструкции "Type", которая достаточно похожа на первую. Это позволяет типизировать пустые и фактические параметры, используя текстуально разные конструкции "Type" без нарушения правил совместимости пустых и фактических параметров. Кроме того, это позволяет специфицировать поля классов информационных объектов, используя одну конструкцию "Type", а соответствующие значения в информационном объекте — используя другую конструкцию "Type", которая достаточно похожа на первую. (Эти примеры не являются исчерпывающими). Рекомендуется, чтобы преимущества, предоставляемые этой свободой, использовались в таких простых случаях, как "SEQUENCE OF INTEGER" или "CHOICE {int INTEGER, id OBJECT IDENTIFIER}", а не в более сложных конструкциях "Type".

F.2 Отображения значений

F.2.1 Лежащей в основе моделью являются типы, такие как неперекрывающиеся контейнеры, в которых находятся значения, и конструкция АСН.1 "Type", каждое появление которой определяет новый, отличный от других, тип (см. рисунки F.1 и F.2). В настоящем приложении устанавливается, когда между такими типами существует отображение значений, позволяющее использовать ссылку на значение одного типа, когда нужна ссылка на значение некоторого другого.

Пример. Рассмотрим

X ::= INTEGER

Y ::= INTEGER

где X и Y — имена ссылок на два разных типа (указатели), но между этими типами существует отображение значений, так что любая ссылка на значение X может использоваться там, где управляет Y (например после DEFAULT).

F.2.2 На множестве всех возможных значений АСН.1 отображение значений относится к паре значений. Все множество отображений значений является математическим отношением. Это отношение обладает следующими свойствами: оно рефлексивно (каждое значение АСН.1 отображается само на себя), симметрично (если, по определению, существует отображение значения x1 в значение x2, то существует и отображение значения x2 в x1) и транзитивно (если существуют отображения значения x1 в x2 и x2 в x3, то существует и отображение значения x1 в x3).

F.2.3 Для любых двух данных типов X1 и X2, рассматриваемых как множества значений, множество отображений значений из X1 в X2 является отношением один к одному, т. е. для всех значений x1 из X1 и x2 из X2, если существует отображение значения x1 в x2, то:

а) нет отображения значения x1 в значение из X2, отличное от x2, и

б) нет отображения значения из X1 (отличного от x1) в значение x2.

F.2.4 Когда существует отображение значений между значениями x1 и x2, ссылка на любое из этих значений автоматически может использоваться для ссылки на другое значение, если это требуется некоторым управляющим типом.

Примечание — Определение отображений между значениями некоторых конструкций "Type" предназначено только для обеспечения гибкости при использовании нотации АСН.1. Существование таких отображений не имеет никакого отношения к тому, что эти два типа несут одну и ту же прикладную семантику, но рекомендуется, чтобы конструкции АСН.1, которые были бы недопустимы без отображения значений, использовались только в тех случаях, когда соответствующие типы действительно выражают одну и ту же прикладную семантику. Часто отображения значения будут существовать в больших спецификациях между типами, которые являются идентичными конструкциями АСН.1, но несут совершенно разную прикладную семантику, и существование таких отображений никогда не должно использоваться при определении допустимости всей спецификации.

F.3 Определения идентичных типов

F.3.1 Понятие "определения идентичных типов" используется для того, чтобы было возможно определить отображения значений между двумя экземплярами "Type", которые либо идентичны, либо достаточно похожи для использования в качестве взаимозаменяемых. Для определения точного смысла выражения "достаточно

похожи" в настоящем разделе специфицирован ряд преобразований, которые применяются к каждому из экземпляров "Type" для приведения их к нормальной форме. Два экземпляра "Type" являются определениями идентичных типов только в том случае, когда их нормальные формы являются идентично упорядоченными списками одних и тех же элементов ACH.1 (см. раздел 11).

F.3.2 Каждый экземпляр "Type" в спецификации ACH.1 является упорядоченным списком элементов, определенных в разделе 11. Нормальная форма получается путем применения преобразований, определенных в F.3.2.1 — F.3.2.4 в указанном порядке.

F.3.2.1 Удаляются все комментарии (см. 11.6).

F.3.2.2 Следующие преобразования не являются рекурсивными и, следовательно, должны применяться только один раз в произвольном порядке.

а) Для каждого целочисленного типа: "NamedNumberList" (см. 18.1), если он есть, переупорядочивается так, чтобы "identifier" были расположены в алфавитном порядке.

б) Для каждого перечислимого типа: к каждому "EnumerationItem" (см. 19.1), который не есть "identifier" (без номера), добавляется номер, как установлено в 19.3; затем "RootEnumeration" переупорядочивается так, чтобы "identifier" были расположены в алфавитном порядке.

в) Для каждого типа "битовая строка" : "NamedBitList" (см. 21.1), если он есть, переупорядочивается так, чтобы "identifier" были расположены в алфавитном порядке.

г) Для каждого значения идентификатора объекта: каждый "ObjIdComponent" преобразуется в соответствующую ему "NumberForm" согласно семантике раздела 31 (см. пример в 31.11).

д) Для типов "последовательность" (см. раздел 24) и "множество" (см. раздел 26): любое расширение вида "ExtensionAndException" или "ExtensionAdditions" вырезается и помещается в конец "ComponentTypeLists"; "OptionalExtensionMarker", если он есть, удаляется.

Когда "TagDefault" есть "IMPLICIT TAGS", то к каждому экземпляру "Tag" добавляется ключевое слово "IMPLICIT TAGS" (см. раздел 30), если только не выполнено одно из следующих условий:

- оно уже есть, или
- присутствует ключевое слово EXPLICIT, или
- тегируемый тип есть CHOICE, или
- это открытый тип.

Когда "TagDefault" есть "AUTOMATIC TAGS", то решение о применении автоматического тегирования принимается в соответствии с 24.2 (автоматическое тегирование будет осуществляться позже).

Примечание — 24.3 и 26.2 устанавливают, что присутствие элемента "Tag" в "ComponentType", который будет вставлен в результате замены "COMPONENTS OF Type", само по себе не предотвращает преобразования автоматического тегирования.

Когда "ExtensionDefault" есть "EXTENSIBILITY IMPLIED", то после "ComponentTypeLists" добавляется многоточие (". . ."), если его не было.

е) Для выборочного типа (см. раздел 28): "RootAlternativeTypeList" переупорядочивается так, чтобы идентификаторы в "NamedType" были расположены в алфавитном порядке. Когда "TagDefault" есть "IMPLICIT TAGS", то к каждому экземпляру "Tag" добавляется ключевое слово "IMPLICIT TAGS" (см. раздел 30), если только не выполнено одно из следующих условий:

- оно уже есть, или
- присутствует ключевое слово EXPLICIT, или
- тегируемый тип есть CHOICE, или
- это открытый тип.

Когда "TagDefault" есть "AUTOMATIC TAGS", то решение о применении автоматического тегирования принимается в соответствии с 24.2 (автоматическое тегирование будет осуществляться позже). Когда "ExtensionDefault" есть "EXTENSIBILITY IMPLIED", то после "AlternativeTypeLists" добавляется многоточие, если его не было.

F.3.2.3 Следующие преобразования должны применяться рекурсивно в указанном порядке до достижения неизменного состояния.

а) Для каждого значения идентификатора объекта (см. 31.3): если определение значения начинается с "DefinedValue", то "DefinedValue" заменяется ее определением.

б) Для типов "последовательность" и "множество": все экземпляры "COMPONENTS OF Type" (см. раздел 24) преобразуются в соответствии с разделами 24 и 26.

в) Для выборочных типов, типов "последовательность" и "множество": если ранее было принято решение о применении автоматического тегирования [см. F.3.2.2, перечисления д) и е)], то оно осуществляется в соответствии с разделами 24, 26 и 28.

г) Для селективного типа: конструкция заменяется выбранной альтернативной в соответствии с разделом 29.

д) Все ссылки на типы заменяются их определениями по следующим правилам:

- если замещающий тип является ссылкой на тип, являющийся преобразованным, то ссылка на тип заменяется специальным элементом, которому не соответствует никакой другой, кроме его самого;
- если замещающий тип является типом "последовательность-из" или "множество-из", то ограничения, следующие за замещаемым типом, если они есть, помещаются перед ключевым словом "OF";
- если замещаемый тип является параметризованным типом или множеством параметризованных значений (см. ИСО/МЭК 8824-4, 8.2), то каждый элемент "DummyReference" заменяется соответствующим "ActualParameter".

e) Все ссылки на значения заменяются их определениями; если замещаемое значение является параметризованным (см. ИСО/МЭК 8824-4, 8.2), то каждый элемент "DummyReference" заменяется соответствующим "ActualParameter".

Примечание — До замены любой ссылки на значение должны быть применены процедуры настоящего приложения для того, чтобы гарантировать, что ссылка на значение идентифицирует,непосредственно или через отображения значений, значение ее управляющего типа.

F.3.2.4 Для типа "множество": "RootComponentTypeList" переупорядочивается так, чтобы "ComponentType" были расположены в алфавитном порядке (от "a" до "z").

Примечание — В 11.9 (bstring), 11.10 (hstring) и 11.11 (cstring) установлено, что новые строки и пропуски в таких элементах значения не имеют. Если два экземпляра таких элементов содержат различающиеся использования новых строк и пропусков, то для целей F.3.3 они трактуются как идентичные.

F.3.3 Если два экземпляра "Type", преобразованные к нормальной форме, являются идентичными списками элементов АСН.1 (см. раздел 11), то они называются определениями идентичных типов за следующим исключением: если "objectclassreference" (см. ГОСТ Р ИСО/МЭК 8824-2, 7.1) встречается в какой-либо нормальной форме "Type", то эти два экземпляра не являются определениями идентичных типов и между ними не существует отображения значений (см. F.4 ниже).

Примечание — Это исключение введено для предотвращения необходимости введения правил преобразования к нормальной форме для элементов синтаксиса, относящихся к нотации информационных объектов, их классов и множеств. Аналогично, не определена нормализация для нотации всех значений и нотации арифметической установки. После доказательства требований для таких спецификаций, они могут быть включены в последующие версии стандарта. Понятия определений идентичных типов и отображений значений введены для обеспечения того, что простые конструкции АСН.1 могут быть использованы либо через имена ссылок, либо копированием текста. Нет необходимости обеспечивать эту возможность для более сложных экземпляров "Type", содержащих классы информационных объектов и пр.

F.4 Спецификация отображения значений

F.4.1 Если два экземпляра "Type" являются определениями идентичных типов по правилам F.3, то существуют отображения между всеми значениями одного типа и соответствующими значениями другого.

F.4.2 Для типа X1, созданного тегированием из некоторого типа X2 (см. раздел 30), отображения значений существуют по определению между всеми членами X1 и соответствующими членами X2.

Примечание — Хотя отображения значений по определению существуют между значениями X1 и X2 из F.4.2 и между значениями X3 и X4 из F.4.3, если такие типы встроены в идентичные в других отношениях, но разные определения типов (таких как определения типов SEQUENCE или CHOICE), получающиеся определения типов (SEQUENCE или CHOICE) не будут определениями идентичных типов и между ними не существуют отображения значений.

F.4.3 Для типа X3, созданного из некоторого управляющего типа X4 выбором значений, конструкцией множества элементов или образованием подтипа, отображения значений существуют по определению между членами нового типа и теми членами управляющего типа, которые были выбраны конструкцией множества элементов или подтипа. Присутствие или отсутствие маркера расширения не влияет на это правило.

F.4.4 В F.5 определены дополнительные отображения значений между некоторыми типами символьных строк.

F.4.5 Отображение значений существует между всеми значениями некоторого типа, определенного как целочисленный тип с поименованными значениями, и любого целочисленного типа, определенного без поименованных значений, с другими поименованными значениями, с другими именами поименованных значений или с другими поименованными значениями и именами одновременно.

Примечание — Существование отображения значений не влияет на требования правила области действия имен и поименованных значений. Они могут использоваться только в области действия, управляемой типом, в котором они определены, или ссылкой на имя этого типа.

F.4.6 Отображение значений существует по определению между всеми значениями любого типа, определенного как битовая строка с поименованными битами, и любого типа, определенного как битовая строка без поименованных битов, с другими поименованными битами, с другими именами поименованных битов или с другими поименованными битами и менами одновременно.

Примечание — Существование отображения значений не влияет на требования правила области действия имен и поименованных битов. Они могут использоваться только в области действия, управляемой типом, в котором они определены, или ссылкой на имя этого типа.

F.5 Дополнительные отображения значений, определенные для типов символьных строк

F.5.1 Имеется две группы ограниченных типов символьных строк: группа A (F.5.2) и B (F.5.3). По определению, отображения значений существуют между всеми типами в группе A и ссылки на их значения могут использоваться, когда они управляются одним из этих типов. Отображения значений никогда не существуют между разными типами в группе B или между какими-либо типами из групп A и B.

F.5.2 Группа A состоит из:

UTF8String
NumericString
PrintableString
IA5String
VisibleString (ISO646String)
UniversalString
BMPString

F.5.3 Группа B состоит из:

TeletexString (T61String)
VideotexString
GraphicString
GeneralString

F.5.4 Отображения значений в группе A определяются отображением значений символьных строк каждого типа в UniversalString и использованием свойства транзитивности отображений значений. Для отображения значений одного из типов группы A в UniversalString строка заменяется на UniversalString той же длины и с отображением каждого символа так, как описано ниже.

F.5.5 Формально множество абстрактных значений UTF8String то же самое, что и множество абстрактных значений UniversalString, но с другим тегом (см. 36.13), и каждое абстрактное значение UTF8String по определению отображается в соответствующее абстрактное значение UniversalString.

F.5.6 Глифы (формы печатных символов), используемые для типов NumericString и PrintableString, распознаются и недвусмысленно отображаются в подмножество глифов, присвоенных первым 128 символам ИСО/МЭК 10646-1. Отображение для этих типов определяется через отображение глифов.

F.5.7 IA5String и VisibleString отображаются в UniversalString путем отображения каждого символа в символ UniversalString, который имеет (32-битовое) значение в кодировании BER для UniversalString идентичное (8-битовому) значению в кодировании BER для IA5String и VisibleString.

F.5.8 BMPString формально является подмножеством UniversalString, и соответствующие абстрактные значения имеют отображения значений.

F.6 Специфичные для типов и значений требования совместимости

В данном разделе понятие отображения значений используется для точной формулировки допустимости некоторых конструкций АСН.1.

F.6.1 Любое появление "Value" в x-нотации с управляющим типом Y идентифицирует y-значение в управляющем типе Y, которое имеет отображение в x-значении, заданное x-нотацией. Требуется, чтобы такое значение существовало.

Например рассмотрим появление x в последней строке следующей записи:

```
X  ::= [0] INTEGER (0 . . 30)
xX ::= 29
Y  ::= [1] INTEGER (25 . . 35)
Z1 ::= Y (x | 30)
```

Эти конструкции АСН.1 допустимы и в последнем присваивании x-нотации x являются указанием x-значения 29 в X и, через отображение значения, идентифицирует y-значение 29 в Y. x-нотация 30 является указанием на y-значение 30 в Y и Z1 является множеством значений 29 и 30. С другой стороны, присваивание

```
Z2 ::= Y (x | 20)
```

недопустимо, так как нет y-значения, на которое может указывать x-нотация 20.

F.6.2 Любое появление "Type" в t-нотации, которое имеет управляющий тип V, идентифицирует все множество значений в управляющем типе V, которые имеют отображения значений в любые значения в "Type" t-нотации. Требуется, чтобы это множество содержало по крайней мере одно значение.

Например рассмотрим появление W в последней строке следующей записи:

```
V  ::= [0] INTEGER (0 . . 30)
W  ::= [1] INTEGER (25 . . 35)
Y  ::= [2] INTEGER (31 . . 35)
Z1 ::= V (W | 24)
```

W привносит значения 25—30 в арифметическое множество, приводя к Z1, имеющему значения 24—30. С другой стороны, присваивание

Z2 : : = V (Y | 24)

недопустимо, так как нет значений в Y, которые отображались в значение в V.

F.6.3 Требуется, чтобы тип любого значения, подставленного в качестве фактического параметра, имел отображение из этого значения в одно из значений в типе, управляющем пустым параметром, и идентифицируется именно это значение в управляющем типе.

F.6.4 Если "Type" подставлен в качестве фактического для пустого параметра, который является пустым параметром множества значений, требуется, чтобы все значения этого "Type" имели отображения в значения в управляющем типе пустого параметра множества значений. Фактический параметр выбирает в управляющем типе полное множество значений, которые имеют отображения в "Type".

F.6.5 При спецификации типа A пустой параметр, который является параметром значения или множества значений, допустим только в том случае, если для всех значений A и для всех использований A в правой части присваивания эти значения A могут быть использованы вместо пустого параметра.

F.7 Примеры

F.7.1 В данном разделе приведены примеры, иллюстрирующие F.3 и F.4

F.7.2 Пример 1

```
X : : = SEQUENCE                X1 : : = SEQUENCE
        {name VisibleString,            {name VisibleString,
         age INTEGER}                    - - комментарий - -
                                         age INTEGER}
X2 : : = [8] SEQUENCE           X3 : : = SEQUENCE
        {name VisibleString,            {name VisibleString,
         age INTEGER}                    age AgeType}
                                        AgeType : : = INTEGER
```

X, X1, X2 и X3 являются определениями идентичных типов. Ни различие в пропусках и комментариях, ни использование ссылки на тип "AgeType" в X3 значения не имеют. Однако если изменить любой идентификатор элемента последовательности, то они перестанут быть идентичными определениями и между ними не будет отображения значений.

F.7.3 Пример 2

```
B : : = SET                     B1 : : = SET
        {name VisibleString,            {age INTEGER,
         age INTEGER}                    name VisibleString}
```

Это определения идентичных типов при условии, что они не находятся в модуле, в заголовке которого стоит "AUTOMATIC TAGS"; в противном случае они не являются определения идентичных типов и между ними не существуют отображения значений. Аналогичные примеры могут быть написаны для CHOICE и ENUMERATED (используя форму "identifier" для "EnumerationItem").

F.7.4 Пример 3

```
C : : = SET                     C1 : : = SET
        {name [0] VisibleString,        {name VisibleString,
         age INTEGER}                    age INTEGER (1 . . 64)}
```

Это не определения идентичных типов, и ни одно из них не является определением идентичного типа с B или B1; также нет отображений значений между C и C1 или между этими типами и B или B1.

F.7.5 Пример 4

```
x  INTEGER { Y (2)} : : = 3
z  INTEGER : : = x
```

Это допустимо, и присваивается значение 3 через отображение, определенное в F.4.5.

F.7.6 Пример 5

```
b1  BIT STRING : : = '101'B
b2  BIT STRING {version1 (0), version2 (1), version3 (2)} : : = b1
```

Это допустимо и b2 присваивается значение {version1, version3}.

F.7.7 Пример 6

С определениями из F.1.1 элементы SEQUENCE вида

X DEFAULT y

допустимы, где X — любой из A, B, C, D, E, F или любой текст справа от присваивания типов этим именам, y — любой из a, b, c, d, e или f, за следующими исключениями: E DEFAULT y недопустимо для всех a, b, c, d, f и C DEFAULT e недопустимо, так как в этих случаях нет отображений значений из ссылки на значение по умолчанию на тип, для которого умолчание устанавливается.

ПРИЛОЖЕНИЕ G
(справочное)

Руководство по модели расширения типа ACH.1

G.1 Обзор

G.1.1 Может случиться так, что тип ACH.1 с течением времени выходит за корень расширения с помощью серии расширений, называемых расширяющими дополнениями.

G.1.2 Тип ACH.1, доступный конкретной реализации, может быть типом корня расширения или типом корня расширения плюс одно или несколько расширяющих дополнений. Каждый тип ACH.1, содержащий расширяющее дополнение, содержит и все ранее определенные расширяющие дополнения.

G.1.3 Говорят, что определения типов ACH.1 в этой последовательности являются связанными расширением (более точное определение понятия "связанные расширением" см. в 3.8.32), и требуется, чтобы правила кодирования применялись к связанным расширением типам таким образом, что если две системы используют два разных связанных расширением типа, то между ними будут успешно переданы те части информации содержимого этих типов, которая является общей для двух систем. Кроме того, требуется, чтобы части, не являющиеся общими для обеих систем, могли быть выделены и в последующем ретранслированы (возможно, третьей стороне) при условии использования того же синтаксиса передачи.

Примечание — Отправитель может использовать тип, который находится в последовательности расширяющих дополнений как раньше, так и позже.

G.1.4 Последовательность типов, полученная добавлениями к корневому типу, называется серией расширений. Для того чтобы правила кодирования могли обеспечить соответствующую передачу связанных расширением типов (которая может потребовать больше битов в строке), такие типы (включая корень расширения) должны быть отмечены синтаксически. Признаком является многоточие (. . .) и называется маркером расширения.

Пример

```
Корень расширения          1-е расширение          2-е расширение          3-е расширение
A ::= SEQUENCE {           A ::= SEQUENCE {        A ::= SEQUENCE {        A ::= SEQUENCE {
    a INTEGER,                 a INTEGER,              a INTEGER,              a INTEGER,
    ...                        ...,                    ...,                    ...,
}                              b BOOLEAN,              b BOOLEAN,              b BOOLEAN,
                               c INTEGER               c INTEGER,              c INTEGER,
                           }                           d SEQUENCE {            d SEQUENCE {
                                                         e INTEGER,              e INTEGER,
                                                         ...,                    ...,
                                                         ...                     g BOOLEAN OPTIONAL,
                                                         f IA5String             h BMPString,
                                                       }                         ...,
                                                   }                             f IA5String
                                                                               }
                                                                           }
```

G.1.5 Все расширяющие дополнения вставляются между парами маркеров расширения. Единственный маркер расширения допустим, если в корне расширения он стоит последним элементом типа; в этом случае принимается, что парный ему маркер расширения стоит непосредственно перед закрывающей фигурной скобкой типа, а расширяющие дополнения вставляются в конец типа.

G.1.6 Тип, имеющий маркер расширения, может быть вложен либо в тип, не имеющий такого маркера, либо в тип в корне расширения, либо в тип расширяющего дополнения. В таких случаях серии расширений трактуются независимо друг от друга и вложенный тип с маркером расширения не влияет на тип, в который он вложен. В любой конкретной конструкции может быть только одна точка вставки расширения (в конце типа, если используется единственный маркер расширения, или непосредственно перед вторым маркером расширения, если используется пара маркеров расширения).

G.1.7 Новое расширяющее дополнение в серии расширений определяется в терминах расширяющей дополнительной группы (из одного или нескольких типов, заключенных между "[[" и "]]") или единственного типа, добавленного в точке вставки расширения. В следующем примере первое расширение определяет расширяющую дополнительную группу, когда b, c должны одновременно присутствовать или отсутствовать в значении типа A. Второе расширение определяет единственный тип компонента d, который может отсутствовать в значении типа A. Третье расширение определяет расширяющую дополнительную группу, в которой компонент h должен присутствовать в значении типа A всякий раз, когда в нем есть новая расширяющая дополнительная группа.

```
        Пример
Корень расширения          1-е расширение          2-е расширение          3-е расширение
A::= SEQUENCE {            A::= SEQUENCE {         A::= SEQUENCE {         A::= SEQUENCE {
    a INTEGER,                 a INTEGER,              a INTEGER,              a INTEGER,
    ...                        ...,                    ...,                    ...,
}                              [[                      [[                      [[

                               b BOOLEAN,              b BOOLEAN,              b BOOLEAN,
                               c INTEGER               c INTEGER,              c INTEGER,
                               ]]                      ]],                     ]],
                           }                           d SEQUENCE {            d SEQUENCE {
                                                         e INTEGER,              e INTEGER,
                                                         ...,                    ...,
                                                         ...                     [[
                                                         f IA5String             g BOOLEAN OPTIONAL,
                                                       }                         h BMPString,
                                                     }                           ]],
                                                                                 ...,
                                                                                 f IA5String
                                                                               }
                                                                             }
```

G.1.8 Хотя обычной практикой должно стать добавление со временем расширяющих дополнений, используемые модель и спецификация АСН.1 не содержат время. Два типа являются связанными расширением, если один может быть "выращен" из другого расширяющими дополнениями. Таким образом, один тип содержит все компоненты другого. Могут быть типы, которые "наращиваются" в противоположном направлении (хотя это мало вероятно). Это может произойти, если тип начинается (во времени) с большого количества расширений, которые последовательно удаляются. Все, что передает АСН.1 и ее правила кодирования, является две спецификации типов связанные или не связанные расширением. Если они связаны, то все правила кодирования АСН.1 должны обеспечивать взаимодействие между их пользователями.

G.1.9 Обычно начинают с некоторого типа и затем решают, хотят ли взаимодействовать с предшествующими версиями, если они были расширены. Если это так, то включают маркер расширения. Затем к типу можно добавить последующие расширяющие дополнения без изменения битов в строке для более ранних значений, но с определенной обработкой расширенных значений ранними системами. Существенно отметить, что добавление маркера расширения к типу, который ранее его не имел, или удаление существующего маркера расширения в общем случае изменит биты в строке и будет препятствовать взаимодействию. Такие изменения в общем случае потребуют изменения версий всех задействованных протоколов.

G.1.10 В таблице G.1 приведены типы АСН.1, которые могут быть типом корня расширения для последовательности расширений АСН.1, и виды единичных расширяющих дополнений, допустимых для этого типа (конечно, кратные расширяющие дополнения могут быть сделаны последовательно или одновременно).

Таблица G.1 — Расширяющие дополнения

Тип корня расширения	Вид расширяющих дополнений
ENUMERATED	Добавление единичного перечисления в конец "AdditionalEnumeration" со значением перечисления большим, чем значение любого добавленного ранее перечисления
SEQUENCE и SET	Добавление единичного типа или расширяющей дополнительной группы в конец "ExtensionAdditionList". "ComponentType", которые являются расширяющими дополнениями (не содержащимися в расширяющей дополнительной группе) не обязательно должны быть отмечены как OPTIONAL или DEFAULT, хотя чаще всего это именно так
CHOICE	Добавление единичного "NamedType" в конец "ExtensionAdditionAlternativesList"
Нотация ограничения	Добавление единичного "AdditionalElementSetSpec" к нотации "ElementSetSpecs"

G.2 Влияние на нумерацию версий

G.2.1 Когда спецификация АСН.1 используется повторно с определениями типов, измененными в терминах определений, связанных расширением типов, то для любых задач эти изменения сами по себе не требуют изменения идентификатора объекта модуля или номера версии протокола.

G.2.2 Может быть так, что по некоторым другим причинам такие изменения должны сопровождаться изменением номера версии, но это не является обязательным требованием.

G.2.3 Напротив, добавление маркера расширения к типу, который ранее его не имел, или добавление компонентов к типу "последовательность" или "множество" без маркера расширения создает новый тип, который не связан расширением со старым, и содержащему его модулю должен быть присвоен новый идентификатор объекта, а с протоколом должен быть связан новый номер версии.

G.3 Требования к правилам кодирования

G.3.1 Абстрактный синтаксис может быть определен как значения единственного типа АСН.1, который является расширяемым. Он содержит все значения, которые могут быть получены добавлением или удалением расширяющих дополнений. Такой абстрактный синтаксис называется связанным расширением абстрактным синтаксисом.

G.3.2 Набор правильно созданных правил кодирования для связанного расширением абстрактного синтаксиса удовлетворяет дополнительным требованиям, установленным в G.3.3 — G.3.5.

Примечание — Правила кодирования АСН.1 удовлетворяют этим требованиям.

G.3.3 Определение процедур для преобразования абстрактного значения в кодирование для передачи и преобразования полученного кодирования в абстрактное значение должно распознавать возможность того, что отправитель и получатель используют разные абстрактные синтаксисы, которые не идентичны, но связаны расширением.

G.3.4 В этом случае правила кодирования должны гарантировать, что когда отправитель имеет спецификацию более раннюю в серии расширений, чем получатель, то значения отправителя должны быть преобразованы таким образом, чтобы получатель мог определить, что расширяющие дополнения отсутствуют.

G.3.5 Правила кодирования должны гарантировать, что когда отправитель имеет спецификацию типа, которая является более поздней в серии расширений, чем спецификация получателя, то передача значений этого типа получателю остается возможной.

ПРИЛОЖЕНИЕ Н
(справочное)

Сводка нотации АСН.1

Следующие элементы определены в разделе 11:

typereference	BEGIN	ISO646String
identifier	BIT	MAX
valuereference	BMPString	MIN
modulereference	BOOLEAN	MINUS-INFINITY
comment	BY	NULL
empty	CHARACTER	NumericString
number	CHOICE	OBJECT
bstring	CLASS	ObjectDescriptor
hstring	COMPONENT	OCTET
cstring	COMPONENTS	OF
":: ="	CONSTRAINED	OPTIONAL
".."	DEFAULT	PDV
"..."	DEFINITIONS	PLUS-INFINITY
"{"	EMBEDDED	PRESENT
"}"	END	PrintableString
"<"	ENUMERATED	PRIVATE
","	EXCEPT	REAL
"."	EXPLICIT	SEQUENCE
"("	EXPORTS	SET
")"	EXTERNAL	SIZE
"["	FALSE	STRING
"]"	FROM	SYNTAX
"-"	GeneralizedTime	T61String
":"	GeneralString	TAGS
";"	GraphicString	TeletexString
"@"	IA5String	TRUE
"|"	TYPE-IDENTIFIER	UNION
"!"	IDENTIFIER	UNIQUE
"^"	IMPLICIT	UNIVERSAL
ABSENT	IMPORTS	UniversalString
ABSTRACT-SYNTAX	INCLUDES	UTCTime
ALL	INSTANCE	VideotexString
APPLICATION	INTEGER	VisibleString
AUTOMATIC	INTERSECTION	WITH

В настоящем стандарте использованы следующие продукции с указанными выше элементами в качестве терминальных символов:

```
ModuleDefinition : : = ModuleIdentifier
                       DEFINITIONS
                       TagDefault
                       ":: ="
                       BEGIN
                       ModuleBody
                       END
ModuleIdentifier : : =  modulereference
                       DefinitiveIdentifier
DefinitiveIdentifier : : = "{" DefinitiveObjIdComponentList "}" | empty
DefinitiveObjIdComponentList : : =
          DefinitiveObjIdComponent      |
          DefinitiveObjIdComponent DefinitiveObjIdComponentList
DefinitiveObjIdComponent : : =
          NameForm        |
          DefinitiveNumberForm          |
          DefinitiveNameAndNumberForm
DefinitiveNumberForm : : = number
```

```
DefinitiveNameAndNumberForm : : = identifier "(" DefinitiveNumberForm")
TagDefault : : =    EXPLICIT TAGS       |
                    IMPLICIT TAGS       |
                    AUTOMATIC TAGS      |
                    empty
ExtensionDefault : : =
                    EXTENSIBILITY IMPLIED          |
                    empty
ModuleBody : : = Exports Imports AssignmentList      |
                    empty
Exports : : =       EXPORTS SymbolsExported ";"    |
                    empty
SymbolsExported : : = SymbolList     |
                        empty
Imports : : =       IMPORTS SymbolsImported ";"      |
                    empty
SymbolsImported : : = SymbolsFromModuleList      |
                        empty
SymbolsFromModuleList : : =
          SymbolsFromModule      |
          SymbolsFromModuleList   SymbolsFromModule
SymbolsFromModule : : SymbolList FROM GlobalModuleReference
GlobalModuleReference : : = modulereference AssignedIdentifier
AssignedIdentifier : : =        ObjectIdentifierValue       |
                                DefinedValue       |
                                empty
SymbolList : : = Symbol | Symbol "," SymbolList
Symbol : : = Reference | ParameterizedReference
Reference : : =
          typereference       |
          valuereference      |
          objectclassreference          |
          objectreference     |
          objectsetreference
AssignmentList : : = Assignment | AssignmentList Assignment
Assignment : : =
          TypeAssignment    |
          ValueAssignment   |
          ValueSetTypeAssignment    |
          ObjectClassAssignment     |
          ObjectAssignment      |
          ObjectSetAssignment |
          ParameterizedAssignment
Externaltypereference : : =
              modulereference
              "."
              typereference
Externalvaluereference : : =
              modulereference
              "."
              valuereference
DefinedType : : =
          Externaltypereference   |
          typereference   |
          ParameterizedType   |
          ParameterizedValueSetType
DefinedValue : : =
          Externalvaluereference   |
          valuereference   |
          ParameterizedValue
```

```
AbsoluteReference : : = "@" GlobalModuleReference
                "."
                ItemSpec
ItemSpec : : =
                typereference     |
                ItemId "." ComponentId
ItemId : : = ItemSpec
ComponentId : : =
                identifier | number | "H"
TypeAssignment : : = typereference
                     ": : ="
                     Type
ValueAssignment : : =          valuereference
                               Type
                               ": : = "
                               Value
ValueSetTypeAssignment : : =    typereference
                                Type
                                ": : = "
                                ValueSet
ValueSet : : = "{" ElementSetSpec "}"
Type : : = BuiltinType | ReferencedType | ConstrainedType
BuiltinType : : =
      BitStringType     |
      BooleanType       |
      CharacterStringType       |
      ChoiceType  |
      EmbeddedPDVType     |
      EnumeratedType      |
      ExternalType    |
      InstanceOfType  |
      Integer  Type |
      NullType    |
      ObjectClassFieldType      |
      ObjectIdentifierType      |
      OctetStringType |
      RealType      |
      SequenceType    |
      SequenceOfType      |
      SetType         |
      SetOfType       |
      TaggedType
NamedType : : = identifier  Type | SelectionType
ReferencedType : : =
      DefinedType |
      UsefulType    |
      SelectionType     |
      TypeFromObject  |
      ValueSetFromObjects
Value : : = BuiltinValue | ReferencedValue
BuiltinValue : : =
      BitStringValue    |
      BooleanValue  |
      CharacterStringValue      |
      ChoiceValue  |
      EmbeddedPDVValue    |
      EnumeratedValue     |
      ExternalValue     |
      InstanceOfValue  |
      IntegerValue    |
      NullValue     |
      ObjectClassFieldValue     |
```

```
        ObjectIdentifierValue     |
        OctetStringValue |
        RealValue     |
        SequenceValue   |
        SequenceOfValue     |
        SetValue    |
        SetOfValue   |
        TaggedValue
ReferencedValue : : =
        DefinedValue     |
        ValueFromObject
NamedValue : : = identifier Value
BooleanType : : = BOOLEAN
BooleanValue : : = TRUE | FALSE
Integer Type : : =
                INTEGER                         |
                INTEGER "{" NamedNumberList "}"
NamedNumberList : : =
                NamedNumber             |
                NamedNumberList  ","  NamedNumber
NamedNumber : : =
                identifier "(" SignedNumber ")"          |
                identifier "(" DefinedValue ")"
SignedNumber : : = number | "-" number
IntegerValue : : = SignedNumber | identifier
EnumeratedType : : =
        ENUMERATED "{" Enumerations "}"
Enumerations : : = RootEnumeration      |
        RootEnumeration "," ". . ."          |
        RootEnumeration "," ". . ." "," AdditionalEnumeration
RootEnumeration : : = Enumeration
AdditionalEnumeration : : = Enumeration
Enumeration : : =
        EnumerationItem | EnumerationItem "," Enumeration
EnumerationItem : : =
        identifier | NamedNumber
EnumeratedValue : : =
        identifier
RealType : : = REAL
RealValue : : =
        NumericRealValue | SpecialRealValue
NumericRealValue : : = 0   |
  SequenceValue - - Значение ассоциированного типа "последовательность"
SpecialRealValue : : =
        PLUS-INFINITY | MINUS-INFINITY
BitStringType : : = BIT STRING | BIT STRING "{" NamedBitList "}"
NamedBitList : : = NamedBit |  NamedBitList "," NamedBit
NamedBit : : = identifier "(" number ")" |
                identifier "(" DefinedValue ")"
BitStringValue : : = bstring | hstring | "{" IdentifierList "}" | "{" "}"
IdentifierList : : = identifier | IdentifierList "," identifier
OctetStringType : : =           OCTET STRING
OctetStringValue : : =          bstring | hstring
NullType : : =          NULL
NullValue : : = NULL
SequenceType : : = SEQUENCE "{" "}"   |
        SEQUENCE "{" ExtensionAndException OptionalExtensionMarker "}"      |
        SEQUENCE "{" ComponentTypeLists "}"
ExtensionAndException : : = ". . ." | ". . . " ExceptionSpec
OptionalExtensionMarker : : = "," ". . . " | empty
ComponentTypeLists : : = RootComponentTypeList     |
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions OptionalExtensionMarker   |
```

```
        RootComponentTypeList "," ExtensionAndException ExtensionAdditions
                                    ExtensionEndMarker "," RootComponentTypeList    |
        ExtensionAndException ExtensionAdditions  ExtensionEndMarker "," RootComponentTypeList
RootComponentTypeList : : = ComponentTypeList
ExtensionEndMarker : : = "," ". . ."
ExtensionAdditions : : = "," ExtensionAdditionList | empty
ExtensionAdditionList : : =  ExtensionAddition |  ExtensionAdditionList "," ExtensionAddition
ExtensionAddition : : = ComponentType | ExtensionAdditionGroup
ExtensionAdditionGroup : : = "[[" ComponentTypeList "]]"
ComponentTypeList : : = ComponentType | ComponentTypeList "," ComponentType
ComponentType : : = NamedType    |
                    NamedType OPTIONAL  |
                    NamedType DEFAULT Value    |
                    COMPONENTS OF Type
SequenceValue : : = "{" Component ValueList "}" | "{" "}"
ComponentValueList : : = NamedValue |  ComponentValueList "," NamedValue
SequenceOfType : : =          SEQUENCE OF Type
SequenceOfValue : : =         "{" ValueList "}" | "{" "}"
ValueList : : = Value | ValueList "," Value
SetType : : = SET "{" "}"      |
          SET "{" ExtensionAndException OptionalExtensionMarker "}"    |
          SET "{" ComponentTypeLists "}"
SetValue : : = "{" ComponentValueList "}" | "{" "}"
SetOfType : : = SET OF Type
SetOfValue : : = "{" ValueList "}" | "{" "}"
ChoiceType : : = CHOICE "{" AlternativeTypeLists "}"
AlternativeTypeLists : : =
        RootAlternativeTypeList   |
        RootAlternativeTypeList "," ExtensionAndException
            ExtensionAdditionAlternatives OptionalExtensionMarker
RootAlternativeTypeList : : = AlternativeTypeList
ExtensionAdditionAlternatives : : =
        "," ExtensionAdditionAlternativesList | empty
ExtensionAdditionAlternativesList : : = ExtensionAdditionAlternative |
        ExtensionAdditionAlternativesList ","
            ExtensionAdditionAlternative
ExtensionAdditionAlternative : : = ExtensionAdditionAlternatives      |
        NamedType
ExtensionAdditionAlternatives : : = "[[" AlternativeTypeList "]]"
AlternativeTypeList : : = NamedType |  AlternativeTypeList "," NamedType
ChoiceValue : : = identifier ":" Value
SelectionType : : = identifier "<" Type
TaggedType : : = Tag Type   |
                 Tag IMPLICIT Type    |
                 Tag EXPLICIT Type
Tag : : =        "[" Class ClassNumber "]"
ClassNumber : : = number | Defined Value
Class : : =      UNIVERSAL    |
                 APPLICATION |
                 PRIVATE        |
                 empty
Tagged Value : : = Value
EmbeddedPDVType : : = EMBEDDED PDV
EmbeddedPDVValue : : = SequenceValue
ExternalType : : = EXTERNAL
ExternalValue : : = SequenceValue
ObjectIdentifierType : : = OBJECT IDENTIFIER
ObjectIdentifierValue : : = "{" ObjIdComponentList "}"      |
                            "{" DefinedValue ObjIdComponentList "}"
Obj IdComponentList : : = ObjIdComponent     |
                          ObjIdComponent ObjIdComponentList
```

```
ObjIdComponent : : =  NameForm        |
                      NumberForm      |
                      NameAndNumberForm
NameForm : : =        identifier
NumberForm : : =      number | DefinedValue
NameAndNumberForm : : =     identifier "(" NumberForm ")"
CharacterStringType : : = RestrictedCharacterStringType      |
                      UnrestrictedCharacterStringType
RestrictedCharacterStringType : : = BMPString |
                      GeneralString  |
                      GraphicString  |
                      IA5String      |
                      ISO646String   |
                      NumericString  |
                      PrintableString |
                      TeletexString  |
                      T61String      |
                      UniversalString |
                      VideotexString |
                      VisibleString
RestrictedCharacterStringValue : : = cstring | CharacterStringList | Quadruple | Tuple
CharacterStringList : : = "{" CharSyms "}"
CharSyms : : = CharsDefn | CharSyms "," CharsDefn
CharsDefn : : = cstring | DefinedValue
Quadruple  : : = "{" Group "," Plane "," Row "," Cell "}"
Group      : : = number
Plane      : : = number
Row        : : = number
Cell       : : = number
Tuple : : = "{" TableColumn "," TableRow "}"
TableColumn : : = number
TableRow : : = number
UnrestrictedCharacterStringType : : = CHARACTER  STRING
CharacterStringValue : : = RestrictedCharacterStringValue  |
                      UnrestrictedCharacterStringValue
UnrestrictedCharacterStringValue : : = SequenceValue
UsefulType : : = typereference
```

Следующие типы символьных строк определены в 36.1:

```
    NumericString              VisibleString
    PrintableString            ISO646String
    TeletexString              IA5String
    T61String                  GraphicString
    VideotexString             GeneralString
    UniversalString            BMPString
```

Следующие полезные типы определены в разделах 39—41:

```
    GeneralizedTime
    UTCTime
    ObjectDescriptor
```

Следующие продукции используются в разделах 42—45:

```
ConstrainedType : : =
      Type Constraint    |
      TypeWithConstraint
TypeWithConstraint : : =
      SET Constraint OF Type    |
      SET SizeConstraint OF Type     |
      SEQUENCE Constraint OF Type    |
      SEQUENCE SizeConstraint OF Type
Constraint : : = "(" ConstraintSpec ExceptionSpec ")"
ConstraintSpec : : =
      SubtypeConstraint    |
      GeneralConstraint
```

```
ExceptionSpec : : = "!" ExceptionIdentification | empty
ExceptionIdentification : : = SignedNumber |
                    DefinedValue          |
                    Type ":" Value
SubtypeConstraint : : = ElementSetSpec
ElementSetSpecs : : =
            RootElementSetSpec  |
            RootElementSetSpec "," "..."    |
            RootElementSetSpec "," "..." "," AdditionalElementSetSpec
RootElementSetSpec : : = ElementSetSpec
AdditionalElementSetSpec : : = ElementSetSpec
ElementSetSpec : : = Unions | ALL Exclusions
Unions : : = Intersections    |
               UElems UnionMark Intersections
UElems : : = Unions
Intersections : : = IntersectionElements | IElems IntersectionMark IntersectionElements
IElems : : = Intersections
IntersectionElements : : = Elements | Elems Exclusions
Elems : : = Elements
Exclusions : : = EXCEPT Elements
UnionMark : : = "|" |    UNION
IntersectionMark : : = "^" INTERSECTION
Elements : : =
    SubtypeElements  |
    ObjectSetElements   |
    "(" ElementSetSpec ")"
SubtypeElements : : =
    SingleValue             |
    ContainedSubtype        |
    ValueRange              |
    PermittedAlphabet       |
    SizeConstraint          |
    TypeConstraint          |
    InnerTypeConstraints
SingleValue : : = Value
ContainedSubtype : : = Includes Type
Includes : : = INCLUDES | empty
ValueRange : : = LowerEndpoint ".." UpperEndpoint
LowerEndpoint : : = LowerEndValue | LowerEndValue "<"
UpperEndpoint : : = UpperEndValue | "<" UpperEndValue
 LowerEndValue : : = Value | MIN
UpperEndValue : : = Value | MAX
SizeConstraint : : = SIZE Constraint
PermittedAlphabet : : = FROM Constraint
TypeConstraint : : = Type
InnerTypeConstraints : : =
    WITH COMPONENT SingleTypeConstraint    |
    WITH COMPONENTS MultipleTypeConstraints
SingleTypeConstraint : : = Constraint
MultipleTypeConstraints : : = FullSpecification | PartialSpecification
FullSpecification : : "{" TypeConstraints "}"
PartialSpecification : : = "{" "..." "," TypeConstraints "}"
TypeConstraints : : =
    NamedConstraint    |
    NamedConstraint   ","   TypeConstraints
NamedConstraint : : =
    identifier  ComponentConstraint
ComponentConstraint : : = ValueConstraint PresenceConstraint
ValueConstraint : : = Constraint | empty
PresenceConstraint : : = PRESENT | ABSENT | OPTIONAL | empty
```

УДК 681.324:006.354 ОКС 35.100.60 П85 ОКСТУ 4002

Ключевые слова: информационная технология, обработка данных, информационный обмен, взаимосвязь открытых систем, уровень представления, спецификации, абстрактная синтаксическая нотация

Редактор *В. П. Огурцов*
Технический редактор *Л. А. Гусева*
Корректор *Н. И. Гаврищук*
Компьютерная верстка *З. И. Мартыновой*

Изд. лиц. № 02354 от 14.07.2000. Сдано в набор 17.09.2001. Подписано в печать 09.11.2001. Усл. печ. л. 12,56. Уч.-изд. л. 12,70
Тираж 400 экз. С 2488. Зак. 2112.

ИПК Издательство стандартов, 107076, Москва, Колодезный пер., 14
http://www.standards.ru e-mail: info@standards.ru
Набрано в Калужской типографии стандартов на ПЭВМ.
Калужская типография стандартов, 248021, Калуга, ул. Московская, 256.
ПЛР № 040138