ASN.1 простыми словами

Юрий Строжевский (http://www.strozhevsky.com)

2012 г.

Введение

Уже на протяжение достаточно большого периода мне приходится иметь дело с ASN.1. Мне посчастливилось работать как в сфере создания криптографических программ, так и в сфере телекоммуникаций. И в той, и в другой сфере изначально крайне активно и повсеместно используется стандарт ASN.1.

Однако и в процессе создания программ криптографической направленности, и в процессе создания программ для телекоммуникационной отрасли я постоянно встречался с одним и тем же мнением - ASN.1 это сложный и не понятный формат, а следовательно для кодирования / декодирования лучше применять сторонние компиляторы (а иногда даже другие стандарты кодирования передаваемой информации).

Одной из причин по которой сложилась ситуация, когда подавляющее большинство разработчиков программ считают стандарт ASN.1 сложным, это отсутствие книг по данному вопросу. Да, не смотря на почтенный возраст данного стандарта, множество свободно распространяемых компиляторов и различных статей, всё ещё крайне мало книг (или даже статей в Интернете) где бы простым и понятным языком, с большим количеством примеров, прояснялись вопросы кодирования простых типов ASN.1.

Исправляя сложившуюся ситуацию данная статья отчасти служит неким пособием, помогающим даже не сталкивавшемуся ранее с этим форматом человеку разобраться в тонкостях кодирования ASN.1. Статья охватывает вопросы исключительно только кодирования простых (не составных) типов - REAL, INTEGER, OBJECT IDENTIFIER, все виды строк, BOOLEAN, NULL, SEQUENCE, SET. В статье приводится подробнейшее объяснение всех тонкостей кодирования для каждого из типов, также приводятся подробные примеры, поясняющий тонкости кодирования для данного типа. В отдельном файле, прилагающемся к данной статье, можно найти код на C++, формирующий все примеры из статьи. Кроме того в этом файле с примерами приводятся дополнительные материалы, не рассмотренные в рамках данной статьи. Все материалы статьи опираются на последний стандарт ASN.1 от 2008 года, все составляющие под-стандарты которого можно скачать одним файлом по ссылке http://www.itu.int/rec/T-REC-X.680-X.693-200811-I/en. Если это специально не оговаривается, то приведенные в статье примеры кодируют типы в стандарте ASN.1 BER (Basic Encoding Rules).

Также упомяну, что буду рад любым (хорошим или критическим) отзывам на эту статью. Отзывы прошу присылать на мою официальную почту yury@strozhevsky.com с пометкой "Статья ASN.1 простыми словами".

В большинстве пособий и книг по ASN.1 изучение кодирования начинается с простейших, не сложных, типов и заканчивается наисложнейшими. В этой статье порядок будет строго противоположный - читателю сначала будет предложено изучить кодирование сложных типов, и только потом постепенно перейдём к изучению простейших. Это позволит однажды усвоив методы кодирования для сложного типа просто и быстро понять методику кодирования более простого.

Глава 1. Общие правила кодирования ASN.1

Первично всё же необходимо пояснить некоторые основы кодирования в формате ASN.1.

Для начала поясним для чего же создавался этот стандарт. В мире существует множество различных компьютеров. И кроме того существует множество стандартов представления данных в этих компьютерах. ASN.1 создавался как некий общий стандарт, позволяющий описывать произвольную информацию, которая бы понималась любым компьютером, имеющим представление об этом стандарте. В стандарте ASN.1 поэтому предъявляются жесткие правила кодирования даже на уровне отдельных битов информации, а также взаимного их расположения. Дополнительно нужно сказать, что стандарт ASN.1 кодирует информацию не в виде текста, а виде двоичных последовательностей. Сейчас уже появились вариации форматов кодирования, позволяющие представлять данные и в виде текста (XML), но обзор этих форматов выходит за рамки данной статьи. Здесь мы рассмотрим только самое сложное - двоичное кодирование (формат ASN.1 BER - Basic Encoding Rules).

Данные закодированные в формате ASN.1 представляют из себя последовательность байт (или "октетов"), которые идут один за другим, без каких либо разрывов. Последовательность закодированную в ASN.1 можно передавать по линиям связи, сохранять в файл - блок закодированной информации в ASN.1 уже содержит необходимое описание его общей длины и содержимого.

Для возможности подобного описания содержащейся в закодированном блоке информации применяется определенная общая структура каждого блока. Каждый блок содержит минимум 3 обязательных части (в отдельных случаях остаются только первые два блока, но эти случаи описываются отдельно):
1. Часть идентификатора блока (до нескольких октетов);
2. Часть общей длины блока (до нескольких октетов);
3. Часть, содержащая собственно значение, которое переносит этот блок (до нескольких октетов);

Кроме этого может быть ещё 4-ая, не обязательная часть - часть октетов окончания значения блока (несколько октетов). Про эту часть будет рассказано несколько позже.

Перейдём к описанию каждой части ASN.1-кодированного блока.

Часть идентификатора блока состоит минимум из одного октета. Формат этого первого октета строго фиксирован.

- биты 8 и 7 (старшие биты, обычно их записывают крайними слева) кодируют так называемый "класс" текущего блока;
  - Биты 8 и 7 равны 00_2 - класс блока UNIVERSAL;
  - Биты 8 и 7 равны 01_2 - класс блока APPLICATION;
  - Биты 8 и 7 равны 10_2 - класс блока CONTEXT-SPECIFIC;
  - Биты 8 и 7 равны 11_2 - класс блока PRIVATE;
- бит 6 должен быть установлен в 0 если текущий блок содержит информацию только об одном значении (примитивное кодирование блока значения) и должен быть установлен в 1, если внутри значения блока содержатся дополнительные ASN.1-кодированные блоки (конструктивное кодирование блока значения);
- биты с 5 по 1 кодируют собственно идентификатор типа для данного блока;

В случае если идентификатор типа для блока находится в диапазоне значений 0-30 идентификационный блок состоит только из одного октета. Если же идентификатор типа для блока имеет значение 31 и выше, то в битах 5-1 выставляются все 1, а в последующих октетах кодируется нужный номер. Номер идентификатора типа кодируется как без знаковое целое, разложенное по основанию 128. В каждом октете, кодирующем

идентификатор типа для блока, старший бит должен быть равен 1, кроме самого крайнего, завершающего октета (способ кодирования полностью совпадает со способом, которым кодируются SID для OBJECT IDENTIFIER, см. ниже).

Часть общей длины блока содержит минимум 1 октет, кодирующий длину значения, которое содержит блок (именно только длину блока, содержащего закодированное значение, а не общую длину всего закодированного блока вместе с идентификатором блока и частью общей длины!). Длина блока в простейшем случае кодируется как без знаковое целое, разложенное по основанию 128. Бит 8 (старший бит) в этом случае является дополнительным флагом. Если общая длина закодированного блока превышает 128, то старший бит первого октета части общей длины блока должен быть установлен в 1, а следующие 7 бит должны кодировать без знаковое целое значение количества последующих октетов, которые и будут кодировать реальную общую длину блока.

Например если общая длина блока равна L = 201 то она будет кодироваться с помощью двух октетов:

1000 0001 (81)
1100 1001 (C9)

Кроме явного задания общей длины блока возможно определять окончание данного блока непосредственно в процессе декодирования блока. Это важно, когда при начальном кодировании блока не ясно, сколько именно октетов он будет содержать (потоковое кодирование). В этом случае первый октет части общей длины блока должен быть равен 80 (старший бит 8 равен 1 и все остальные биты равны 0). Окончание всего блока определяется по наличию в блоке значения двух последовательно идущих октетов 00 00.

Глава 2. Кодирование типа REAL

Общее описание типа:

- Класс тэга - UNIVERSAL (00);
- Номер тэга - 9;
- Форма кодирования значения - примитивная (не конструктивная форма);

Для начала немного теории, касающейся собственно чисел с плавающей запятой. Числа с плавающей запятой обычно представляют состоящими из трёх частей: мантиссы, основания и экспоненты. Более просто это можно объяснить с помощью формулы: REAL = (мантисса)*(основание)$^{(экспонента)}$. Если по этой формуле представлять обычные десятичные числа, то получится REAL = (мантисса)*10$^{(экспонента)}$. Так как в ASN.1 и мантисса и экспонента могут быть как положительными, так и отрицательными, то возможно как представление сколь угодно больших и сколь угодно маленьких значений, с произвольным знаком.

В отличие от обычного, машинного, представления чисел с плавающей запятой (IEEE 754) в ASN.1 тип REAL практически не ограничен по размеру как мантиссы (мантисса может состоять из практически не ограниченного числа октетов и представлять сколь угодно большое число), так и по размеру экспоненты (значение экспоненты также может состоять из произвольного количества октетов). Ограничения при кодировании накладываются только на значение "основания": в качестве "основания" могут выбраны только числа 10, 2, 8 или 16.

Для кодирования типа REAL применяются следующие три основных блока:

1. Служебный информационный октет;
2. Значение экспоненты числа;
3. Значение мантиссы числа;

В служебной информационном октете содержится следующая информация:

- Возможные комбинации битов 8 и 7 (крайние слева):
  - Бит 8 = 1 - применяется двоичное кодирование (по одному из оснований 2, 8 или 16);
  - Бит 8 = 0 и бит 7 = 0 - применяется десятичное кодирование (фактически кодирование строкового стандартного представления числа, см. далее);
  - Бит 8 = 0 и бит 7 = 1 - закодированное значение является "специальным значением" (NaN, INFINITE etc.) или закодированное значение кодирует "-0";
- Бит 7 установлен в 0 когда кодируемое число положительно, и установлен в 1 когда кодируемое число отрицательно;
- Комбинация битов 6 и 5 определяет базу двоичного кодирования:
  - 00 - кодируемое число разложено по основанию 2;
  - 01 - кодируемое число разложено по основанию 8;
  - 10 - кодируемое число разложено по основанию 16;
  - 11 - зарезервировано для будущих возможных изменений;
- Биты 4 и 3 кодируют значение "scaling factor" (F, см. далее) в двоичном коде;
- Биты 2 и 1 кодируют формат представления экспоненты в закодированном числе:
  - 00 - следующий октет представляет собой единственный октет, кодирующий значение экспоненты;
  - 01 - следующие два октета кодируют значение экспоненты;
  - 10 - следующие три октета кодируют значение экспоненты;
  - 11 - следующий октет содержит количество последующих октетов, кодирующих значение экспоненты (количество октетов кодируется как обычное число без знака (допускаются только положительные значения, естественно), а последующие октеты кодируют значение экспоненты;

Значение экспоненты числа кодируется целым числом, состоящим из произвольного количества октетов. Целые числа (как отрицательные, так и положительные) в ASN.1 кодируются с помощью так называемого "дополнительного кода". Ниже я дам подробное описание алгоритма кодирования для положительных и отрицательных целых чисел.

Положительные целые числа в ASN.1 представляют собой последовательность "индексов" при соответствующих степенях разложения по основанию 256. То есть целое число, представленное в обычном десятичном формате, сначала раскладывается по основанию 256, а потом индексы при соответствующих степенях 256 записываются в качестве кодирующих октетов. Для наглядного примера возьмём число 32639. Данное число разлагается по основанию 256 как: $32639_{10} = 127*256^1 + 127*256^0$. Следовательно коэффициенты при соответствующих степенях 256 будут равны (127, 127). Представляя десятичное значение 127 в виде последовательности битов получаем: 127 = 0111 1111, или представляя каждую группу из четырех битов в качестве числа от 0 до F получаем: $(127)_{10} = (0111\ 1111)_2 = 7F_{256}$. Таким образом начальное число 32639 будет кодироваться последовательностью из двух октетов $(7F\ 7F)_{256}$.
Рассмотренным выше способом можно закодировать сколь угодно большое целое положительное число. Однако как быть с кодированием отрицательных целых значений? Именно для кодирования отрицательных целых применяется специальная процедура кодирования значений.

Для примера опять возьмем число 32639, но теперь пусть оно будет отрицательным (-32639). Кодирование отрицательных целых построено так, что на самом деле кодируется не одно, а два целых значения - одно основное значение и другое целое значение, которое нужно вычесть из основного значения. То есть при декодировании для получения закодированного отрицательного числа просто вычислить результат (x - y). Как видно из этой простейшей формулы если значение "x" меньше, чем значение "y" то результат будет меньше нуля (то есть отрицательное число).
Вышеупомянутые два числа (основное число и число, которое надо вычесть из основного) формируются по следующим правилам:

- Пусть закодированное в ASN.1 число состоит из последовательности из N бит;
- Тогда число, которое надо вычесть из основного числа, образуется как число также состоящее из N бит, но где все биты, кроме самого старшего (крайний левый бит), установлены в 0;
- Основное число также состоит из N бит, но в нем самый старший бит установлен в 0. Значения всех остальных битов полностью соответствуют соответствующим битам из изначально закодированного числа (остаются неизменными);

Перейдём к кодированию конкретного числа из примера (-32639). Так как число, которое надо вычесть из основного, должно быть больше основного числа, то кодирование отрицательных целых чисел начинается именно с выбора этого вычитаемого. Так как по правилам это вычитаемое должно разлагаться по основанию 256 так, чтобы все биты, представляющие индексы при соответствующих степенях 256, были равны 0 кроме первого бита, то ряд возможных вычитаемых представляет собой лидирующий октет 80 (1000 0000) и какое-то количество октетов 00, следующих за ним. То есть в качестве вычитаемых могут использоваться: 80_{256} (128_{10}), $(\ 80\ 00\ )_{256}$ (32768_{10}), $(\ 80\ 00\ 00\ )_{256}$ (8388608_{10}) и т.п. Для кодирования нашего числа "-32639" выберем первое подходящее вычитаемое, большее кодируемого числа по модулю (то есть большее чем число 32639). Ближайшее такое число равно 32768 ($80\ 00_{256}$).
Теперь необходимо получить значение основного числа. Для этого надо опять решить простейшую формулу: x - 32768 = -32629. Решая уравнение получаем значение x = 129 = $129*256^0$, следовательно число 129 кодируется одним байтом 81_{256}. Так как если более внимательно рассматривать правила то можно понять, что количество бит в основном и вычитаемом числах должно быть равно. Количество бит в вычитаемом равно 16. В то же время количество бит в основном числе равно всего 8. Для увеличения числа бит в основном числе просто добавим не значащие нули для старших бит. Тогда получим 129 = $0*256^1 + 129*256^0$, а следовательно основное число будет кодироваться двумя октетами как $(00\ 81)_{256}$. Теперь устанавливая первый бит в 1 для полученного двух октетного основного числа получаем окончательное число, которое кодирует "-32639". Это число будет кодироваться двумя октетами $(\ 80\ 81\ )_{256}$. Ещё раз - основное число образуется из всех битов закодированного числа, кроме самого старшего бита (получаем что основное число у нас кодируется $(\ 00\ 81\ )_{256}$), а вычитаемое число образуется только из одного первого бита, установленного в 1, и всех остальных бит, установленных в 0 (получаем, что вычитаемое число у нас кодируется как $(\ 80\ 00\ )_{256}$).

А теперь приятная информация - в современных компьютерных системах целые числа (как положительные, так и отрицательные) автоматически кодируются и хранятся именно в том формате, который и был описан выше. То есть для кодирования целых чисел в ASN.1 не нужно выполнять вообще никаких действий - просто нужно сохранить их байт за байтом и всё.

Значение мантиссы числа представляет собой всегда без знаковое целое. То есть мантисса числа, кодированного в ASN.1, всегда является положительным числом. Для того чтобы

кодировать отрицательные числа с плавающей точкой в ASN.1 предусмотрен отдельный бит (бит 7) в служебном октете (см. выше).
Мантисса кодируется как последовательность байт представляющих собой коэффициенты разложения начального числа по основанию 256. То есть если мантисса числа в десятичном виде равна 32639 то значит закодированное число будет состоять из двух октетов 7F 7F ($32639_{10} = 127*256^1 + 127*256^0 = 7F*FF^1 + 7F*FF^0$).

Примеры кодирования чисел REAL в ASN.1 в двоичном представлении:

Для примера возьмём число 0.15625. Для начала закодируем его в двоичном представлении по основанию 2. Коэффициенты разложения этого числа по основанию 2 будут находится как: $0.15625_{10} = 1*2^{-3} + 1*2^{-5}$. То есть мантисса для нашего тестового числа будет иметь значение $M = 101_2$, а значение экспоненты будет равно -5. Служебный октет для этого числа будет $1000\ 0000_2 = 80_{256}$. Значение экспоненты будет кодироваться одним октетом: -5 = 123 - 128 и следовательно основное число будет равно $123_{10} = 7B_{256}$, а вычитаемое число равно $128_{10} = 80_{256}$. Тогда окончательный октет, кодирующий число -5, будет равен FB_{256}. Значение мантиссы также кодируется одним октетом: $101_2 = 05_{256}$. Теперь нам известны все части блока, кодирующего значение 0.15625 в двоичном коде по основанию 2 и весь кодирующий блок будет состоять из трёх октетов $(80\ FB\ 05)_{256}$.

Теперь закодируем это же число 0.15625, но уже по основанию 8. Коэффициенты разложения этого числа по основанию 8 будут находится как: $0.15625_{10} = 1*8^{-1} + 2*8^{-2}$. То есть мантисса для нашего тестового числа будет иметь значение $M = 12_8 = (001\ 010)_2$ (при кодировании числа в 8-миричной системе для каждого значения требуется три отдельных бита). Значение экспоненты будет равно -2. Служебный октет для этого числа будет $1001\ 0000_2 = 90_{256}$. Значение экспоненты будет кодироваться одним октетом, где основное и вычитаемое число находятся из формулы: -2 = 126 - 128. Следовательно октет, кодирующий значение экспоненты -2, будет FE_{256}. Значение мантиссы числа будет также кодироваться одним октетом $0A_{256}$.

В этом примере разложим число 0.15625 по основанию 16. Коэффициенты этого разложения будут находится как: $0.15626_{10} = 2*16^{-1} + 8*16^{-2}$. Следовательно получаем выражение для мантиссы $M = 28_{16} = (0010\ 1000)_2$ и значение экспоненты E = -2. Теперь поставим дополнительное условие: значение мантиссы должно быть "нормализовано", то есть не должно содержать нулей в младших разрядах числа (также это требование зачастую звучит как "мантисса должна быть нечетной", так как если крайний младший бит равен 1, то всё число получается нечетным ввиду того, что к степеням двойки добавляется $1*2^0$). Как может быть выполнено подобное условие "нормализации"? Очевидно, что основной способ - изменение значения экспоненты числа, сдвигающее плавающую точку. В случае использования разложения по основанию 2 всё представляется простым - изменение значения экспоненты на 1 сдвигает плавающую точку (или добавляет/удаляет нули в младших разрядах мантиссы) ровно на одну позицию. Однако в случае использования разложения по основаниям 8 и 16 получаем, что изменение значения экспоненты на 1 сдвигает плавающую точку в мантиссе сразу на 3 и 4 бита соответственно (так как в случае разложения по основанию 8 для представления числа требуется 3 бита, а в случае разложения по основанию 16 для представления числа требуется 4 бита). Следовательно далеко не всегда полученное для разложения по основаниям 8 и 16 значение мантиссы может быть "нормализовано" просто изменением значения экспоненты. Для более "тонкой настройки" возможности сдвига плавающей точки в мантиссе был введен дополнительный множитель: умножающий фактор, F. Умножающий фактор сдвигает плавающую точку в мантиссе вправо (или добавляет необходимое количество нулевых бит справа от числа). Для

этого перед декодированием значение мантиссы получается как результат умножения $M = N * 2^F$. Общеизвестно, что умножение целого числа на 2 равноценно битовому сдвигу влево на 1 бит. Соответственно умножение на 2^F равноценно битовому сдвигу влево на F бит. Таким образом получаем следующий процесс кодирования/декодирования мантиссы при предъявлении требования её нормализации:

a. Пусть дана мантисса 0010 1000;
b. При кодировании "нормализуем" её (или сдвигаем вправо на 3 бита), получая 0000 0101, одновременно устанавливая значение умножающего фактора F = 3;
c. При декодировании умножаем закодированное значение мантиссы на 2^F, чем сдвигаем закодированную мантиссу обратно на F = 3 бита влево;

Следовательно все число с плавающей точкой из нашего примера (при условии "нормализации" мантиссы) будет кодироваться следующей последовательностью октетов:

$(AC\ FE\ 05)_{256}$

Кроме кодирования всех частей числа с плавающей точкой в виде двоичного представления в разложении по различным степеням двойки дополнительно есть прекрасная возможность представлять подобные числа в ASN.1 в обычном строковом виде, в каком мы обычно и видим такие числа. В этом случае считается, что число кодируется с основанием 10.

При кодировании по основанию 10 дополнительно вводится понятие "форм представления числа". Всего таких форм 3 (формы NR1, NR2 и NR3) и описываются они в отдельном стандарте ISO 6093. Так как этот стандарт является платным, то для ознакомления с формами представления чисел можно порекомендовать "предка" ISO 6093 - стандарт ECMA-63, который легко может быть найден в Интернете.

При кодировании числа с плавающей точкой в представлении разложения по основанию 10 в служебном информационном октете указывается код формы представления числа (01, 02 или 03 для соответствующих форм), а сразу после служебного информационного октета указываются коды символов, представляющих кодированное число. Разрешены следующие коды символов:

1. Символы, обозначающие цифры 0-9 (коды 30_{256} - 39_{256} соответственно);
2. Пробел (код 20_{256});
3. Разделительный символ "." (код $2E_{256}$);
4. Разделительный символ "," (код $2C_{256}$);
5. Символ представление экспоненты "E" (код 45_{256}), либо тот же символ представления экспоненты, но в виде "e" (код 65_{256});
6. Символ "-" (код $2D_{256}$);
7. Символ "+" (код $2B_{256}$);

Все остальные символы запрещены к кодированию (при декодировании символов, отличных от приведенных выше, декодер ASN.1 обязан выдать ошибку).

Примеры кодирования числа с плавающей точкой в десятичной форме:

Для примера закодируем обычное число 1. В случае представления в форме NR1 число будет кодироваться строкой "1" (или "+1"). В случае представления числа в форме NR2 число уже должно быть закодировано с указанием разделительного символа, поэтому все представленные ниже строки равноценны:

1. "1,"
2. "+1.0"
3. "1,000000"

4. " 1.0" (в начале строки может присутствовать неограниченное количество пробелов)

Теперь представим 1 в форме NR3. Здесь уже обязательно применение как разделительного символа, так и символа экспоненты. В форме NR3 по стандарту 1 может представляться в виде "+1,0E+0" ("1.0E+0" в случае разделительного символа "."), то есть значение экспоненты всегда должно быть нулевым.

Кроме обычных чисел ASN.1 позволяет кодировать также и ряд "специальных" чисел:

- PLUS-INFINITY (плюс бесконечность);
- MINUS-INFINITY (минус бесконечность);
- NOT-A-NUMBER (так называемое "не-число");
- minus zero (для возможности кодирования "-0");

Все специальные числа кодируются только одним служебным информационным октетом, без указания октетов для экспоненты и мантиссы:

- PLUS-INFINITY - 40_{256};
- MINUS-INFINITY - 41_{256};
- NOT-A-NUMBER - 42_{256};
- minus zero - 43_{256};

Глава 3. Кодирование типа OBJECT IDENTIFIER

Общее описание типа:

- Класс тэга - UNIVERSAL (00);
- Номер тэга - 6;
- Форма кодирования значения - примитивная (не конструктивная форма);

Сам по себе OBJECT IDENTIFIER представляет собой последовательность целых без знаковых чисел. В стандартном представлении для OBJECT IDENTIFIER элементы этой последовательности представляются в виде строки, разделённые знаком ".". Примеры возможных OBJECT IDENTIFIER: 0.1.1, 1.1.1, 2.1234.1234.1234.1234.
Кодирование OBJECT IDENTIFIER (OID) состоит из последовательного кодирования всех составляющих данный OID без знаковых целых (SID - sub identifier).

Для уменьшения общего размера закодированного OID к первым двум SID применяются следующие правила:

- Первое без знаковое целое (SID1) должно быть или 0, или 1, или 2. Другие значения недопустимы. SID1 исключается из процесса кодирования, вместо этого значение SID1 вычисляется естественным путём при декодировании SID2 (см. ниже);
- Перед кодированием второго SID к нему применяется следующая формула: SID2 = SID1*40 + SID2;
    - Для гарантированного определения SID1 из результата полученного выражения для SID2 к SID2 предъявляются следующие требования:
        - Если SID1 = 0 или SID1 = 1 то значение SID2 должно лежать в диапазоне от 0 (включительно) до 39 (включительно);
        - Если SID1 = 2, то SID2 может выражаться произвольным целым без знаковым числом;
    - Примеры кодирования первых двух октетов:
        - "0.39" кодируется одним целым числом 0*40 + 39 = 39;

- "1.0" кодируется одним целым числом 1*40 + 0 = 40;
- "1.39" кодируется одним целым числом 1*40 + 39 = 79;
- "2.0" кодируется одним целым числом 2*40 + 0 = 80;
- "2.39" кодируется одним целым числом 2*40 + 39 = 119;
- "2.339" кодируется одним целым числом 2*40 + 339 = 419;

Каждый SID кодируется независимо от остальных. Все SID кодируются друг за другом без дополнительных разделителей. Правила кодирования SID одинаковы для всех SID кроме первых двух (см. выше).

Нужно отметить, что теоретически каждый SID может иметь сколько угодно большое целое значение. Следовательно каждый отдельный SID может кодироваться произвольным, сколь угодно большим, количеством октетов. Обычно для кодирования переменного количества октетов применяется дополнительное кодирование количества этих октетов (перед последовательностью закодированных октетов кодируется октет, содержащий количество закодированных октетов). Однако в случае кодирования SID применяется другой подход - самый старший бит каждого из октетов, кодирующих отдельный SID, является своеобразным флагом, по которому можно судить последний ли это октет для данного SID или нет. То есть так как один SID может быть закодирован с помощью более чем одного октета (в случае большого целого числа), то применяется следующее правило: во всех октетах, кодирующих значение SID, кроме последнего (младшего) октета значение самого старшего бита должно быть установлено в 1. Таким образом получается, что значащими для кодирования SID битами в 8-ми битовом октете являются только младшие 7 бит, старший бит в каждом октете используется как флаг, указывающий на последний октет. В связи с этим перед кодированием каждый SID переводится из десятичной формы в форму разложения по основанию 128 (группируется по 7 бит).

Примеры кодирования SID

- SID = 643_{10} = $5*128^1 + 3*128^0$ = (05 03)$_{128}$. Так как все октеты, кроме самого младшего, должны иметь установленный старший бит, то SID равный 643 кодируется в ASN.1 с помощью двух октетов (85 03)$_{256}$;
- SID = 113549_{10} = $6*128^2 + 119*128^1 + 13*128^0$ = (06 77 0D)$_{128}$. Так как все октеты, кроме самого младшего, должны иметь установленный старший бит, то SID равный 113549 кодируется в ASN.1 с помощью трех октетов (86 F7 0D)$_{256}$;
- SID = 49152_{10} = $3*128^2$. В этом случае в разложении отсутствуют младшие степени числа 128. Но так как при кодировании SID нигде не сохраняется значение экспоненты числа, то для нужд кодирования 49152 должно представляться в виде ($3*128^2 + 0*128^1 + 0*128^0$) = (03 00 00)$_{128}$ (то есть закодированное число должно всегда состоять из числа октетов больших на 1 старшей степени числа 128 при разложении). Так как все октеты, кроме самого младшего, должны иметь установленный старший бит, то SID равный 49152 кодируется в ASN.1 с помощью трех октетов (83 80 00)$_{256}$;

При кодировании SID должно использоваться минимальное количество октетов. То есть SID = 643 не должен быть представлен как ($0*128^2 + 5*128^1 + 3*128^0$) = (00 05 03)$_{256}$ и, следовательно, не должен быть закодирован как (80 85 03)$_{256}$. Простейшая проверка на правильность кодирования SID - первый октет закодированного SID не должен быть равен 80_{256}.

Глава 4. Кодирование типа INTEGER

Общее описание типа:

- Класс тэга - UNIVERSAL (00);
- Номер тэга - 2;
- Форма кодирования значения - примитивная (не конструктивная форма);

Вопросы кодирования целых (INTEGER) уже рассматривался ранее (в главе про кодирование типа REAL). Однако еще раз напомним основные особенности этого кодирования.

В ASN.1 могут быть закодированы как положительные, так и отрицательные числа. Каждое целое число практически не ограничено в своей величине (то есть в ASN.1 могут быть закодированы сколь угодно большие (по модулю) целые числа).

Каждое целое число кодируется последовательностью октетов, каждый октет представляет собой 8 бит информации. Каждый октет представляет собой "вес" перед соответствующей степенью числа 256, участвующий в разложении кодируемого числа по основанию 256. То есть для кодирования числа исходное число сначала разлагается по основанию 256, и затем значения "весов" перед соответствующими степенями 256 кодируются в качестве октетов. Например число 8388607 будет кодироваться по следующей схеме:

- Разложим число по основанию 256: $8388607_{10} = 127*256^2 + 255*256^1 + 255*256^0$;
- Получаем, что "веса" при соответствующих степенях 256 будут: 127, 255 и 255;
- Переводим каждое число в последовательность бит, а затем кодируем эту последовательность бит, группируя в группы по 4 бита. Получаем следующие значения для "весов": 7F FF FF;

Кодирование отрицательных целых значений осуществляется по отдельным правилам. Фактически в закодированном отрицательном целом хранится не одно, а два целых числа: основное число и число, которое нужно вычесть из основного числа, чтобы при декодировании получить изначально закодированное отрицательное число. То есть при декодировании изначальное отрицательное число получается по формуле: N = X - Y, где X - основное число, а Y - вычитаемое число. Не трудно понять, что для того, чтобы N получилось отрицательным, необходимо чтобы выполнялось условие Y > X.

Подробные правила формирования как основного, так и вычитаемого числа уже были описаны ранее, поэтому отсылаю читателя к главе о кодировании типа REAL.

Примеры кодирования целых чисел:

- Пусть имеем уже закодированное число $80_{256} = 128*256^0 = 128_{10} = (1000\ 0000)_2$. Здесь основное число образуется битами 1-7 (младшими, крайними справа) и равно 0, вычитаемое образуется путём маскирования всех битов, кроме старшего и равно $(1000\ 0000)_2 = 80_{256} = 128_{10}$. Следовательно закодированное число равно 0 - 128 = -128;
- Кодирование положительного значения +128 осуществляется с помощью последовательность октетов $0*256^1 + 128*256^0 = (00\ 80)_{256}$, то есть добавляется старший нулевой октет. Фактически здесь тоже можно вычислить как основное число, так и вычитаемое число: основное число будет равно 128, а вычитаемое будет равно 0;

- Закодируем число (-8388607). Число $8388607_{10} = 127*256^2 + 255*256^1 + 255*256^0 = (\ 7F\ FF\ FF\ )_{256}$. Так как здесь не установлен старший бит, то вычитаемое число будет равно $(\ 80\ 00\ 00\ )_{256} = 128*256^2 + 0*256^1 + 0*256^0 = 8388608_{10}$. Следовательно основное число может быть получено в результате решения уравнения: x - 8388608 = -8388607. То есть получаем x = 1 и вроде бы это число раскладывается по основанию 256 просто в 0x01. Однако нам необходимо, чтобы вычитаемое число было $(80\ 00\ 00)_{256}$, а следовательно основное число должно выражаться в виде $(00\ 00\ 01)_{256}$. Устанавливая первый бит основного числа получаем, что окончательное кодирование для числа (-8388607) осуществляется тремя октетами $(80\ 00\ 01)_{256}$;

Фактически к кодируемому целому числу предъявляется требование: старшие 9 бит закодированного числа не должны все быть равны 1, и не должны все быть равны 0 (старшие 9 бит закодированного числа не должны быть равны между собой). Если первые 9 бит закодированного числа равны 0, то значит без ущерба для закодированного значения старший октет (старшие 8 бит) можно отбросить (при декодировании он бы просто добавил слагаемое $0*256^n$, что никак не повлияет на значение числа). Если же старшие 9 бит закодированного числа равны 1 то закодированное число отрицательное и может быть перекодировано с применением меньшего числа октетов (при кодировании были добавлены лишние нулевые октеты в вычитаемое число).

Пример добавления лишних нулевых октетов в вычитаемое число. Возьмем для кодирования число (-128). Положим вычитаемое число будет равно: $(80\ 00)_{256} = 32768_{10}$, и основное значение вычисляется после решения уравнения (x - 32768) = -128. Следовательно $x = 32640_{10} = (\ 7F\ 80\ )_{256}$. Устанавливая старший бит получаем окончательное кодирование $(FF\ 80)_{256}$ (здесь равны между собой 9 старших бит). Но ранее мы уже кодировали число (-128) и знаем, что кодирование этого числа в ASN.1 может быть представлено в более короткой форме - в виде только одного октета $(80)_{256}$;

Ну и в конце этой главы ещё раз напомним читателю, что в современных компьютерных системах кодирование целых чисел уже (автоматически) осуществляется по выше описанным правилам. Правда с одной оговоркой: для кодирования отрицательных чисел все-таки применяется "добавление лишних нулевых октетов в вычитаемое число" (см. предыдущий параграф). То есть если количество октетов (байт) для хранения целого числа составляет 4, то число (-128) будет закодировано в виде FF FF FF 80 (то есть при кодировании было применено "вычитаемое число" равное $(80\ 00\ 00\ 00)_{256} = 2147483648$).

Глава 5. Кодирование строковых значений

В ASN.1 достаточно широкий выбор строковых типов. Вот их полный перечень:

- NumericString;
- PrintableString;
- TeletextString (тип полностью совпадает с T61String);
- VideotexString;
- VisibleString;
- IA5String;
- GraphicString;
- GeneralString;
- UniversalString;
- BMPString;
- UTF8String;

Некоторые из них уже являются устаревшими и не применяемыми типами строк (например VideotexString). Каждый тип строк описывает некий набор символов, которые можно применять в строках с данным типом. Дополнительно в используемых строках также могут применяться так называемые "управляющие последовательности", позволяющие автоматически настраивать обработку отдельно выделенной строки на поддерживающих эту возможность терминалах. Тот, кто программировал на C\C++ наверняка часто использовал "управляющую последовательность" \n - перевод строки. На самом деле управляющих последовательностей очень много уже в стандарте, кроме того можно создавать собственные управляющие последовательности, которые будут обрабатываться только на специализированных терминалах (например переключать цвет выделенных строк на красный, подчеркивать строки и т.п.).

Наиболее продвинутыми и современными форматами являются форматы строк, основанные на стандарте Unicode. Основу стандарта составляет документ ISO 10646.

Тип UniversalString (класс тэга UNIVERSAL, номер тэга 28, форма кодирования - примитивная). Кодирует Unicode строки, где каждый символ кодируется 4-мя байтами (октетами).

Тип BMPString (класс тэга UNIVERSAL, номер тэга 30, форма кодирования - примитивная). Подмножество символов Unicode, каждый символ кодируется всегда 2-мя байтами (октетами).

Тип UTF8String (класс тэга UNIVERSAL, номер тэга 12, форма кодирования - примитивная). Представление символов Unicode, но с дополнительной обработкой, позволяющей кодировать каждый символ в последовательность байт переменной длины (от 1 до 7-ми байт).

Глава 6. Кодирование даты и времени

Все типы описывающие дату и время в ASN.1 являются обыкновенными UTF-8 строками, где значения для года, месяца и так далее кодируется в определенном формате. Форматы для представления каждого типа описываются в свободно доступном стандарте ISO 8601.

Тип UTCTime. Простейший временной тип. Может кодировать только дату и UTC (Universal Coordinated Time) время. Формат строки, кодирующей UTCTime, может иметь либо вид YYMMDDHHMMSSZ (где YY - две последние цифры года, MM - две цифры месяца, DD - две цифры дня, HH - две цифры часа (формат 24-х часовой), MM - две цифры минут, SS - две цифры секунд), либо может также кодировать UTC время вместе с относительным смещением для соответствующего часового пояса в виде YYMMDDHHMMSS+hhmm (или YYMMDDHHMMSS-hhmm).

Тип GeneralizedTime. Расширение типа UTCTime, использующее уже 4 цифры для обозначения года, плюс тип GeneralizedTime позволяет использовать дробные значения для любой из временных составляющих (часа, минуты или секунды). Соответственно форматы могут быть следующими:

- YYYYMMDDHHMMSSZ;
- YYYYMMDDHHMMSS+hhmm;
- YYYYMMDDHHMMSS-hhmm;
- YYYYMMDDHHMMSS.nn (количество знаков после точки не ограничено);

- YYYYMMDDHHMM.nn (количество знаков после точки не ограничено);
- YYYYMMDDHH.nn (количество знаков после точки не ограничено);

Все описанные ниже типы являются новыми для последней версии стандарта X.680:2008.

Тип TIME. Описывает только время в формата HH:MM:SS, либо в формате HHMMSS.

Тип TIME-OF-DAY. То же самое, что и тип TIME, за исключением того, что формат может быть только HHMMSS.

Тип DATE. Описывает только дату в формате YYYYMMDD.

Тип DATE-TIME. Описывает как дату, так и время для этой даты. Формат представления YYYYMMDDHHMMSS. В случае если значение крайних фракций времени равно 0 (то есть например 0 секунд), то нулевое значение может быть исключено (строка сокращена).

Тип DURATION. Описывает разность между двумя временными промежутками. Формат представления nnYnnMnnDTnnHnnMnnS.

Глава 7. Кодирование последовательностей бит (битовых строк)

В разряд типов, кодирующих битовые строки, я отношу два типа - BIT STRING и OCTET STRING.

Тип BIT STRING предназначен для хранения последовательностей наименьших блоков информации - битов. Фактически никакого кодирования не происходит - в блок значения для закодированного значения помещается блок кодируемого значения. За одним только исключением - в первом октете закодированного значения хранится число не используемых бит. Значения количества не используемых бит может изменяться от 0 до 7. Не используемые биты располагаются в битовой строке крайними справа. То есть если закодирована последовательность бит 0000 1111 = 0F, и число не используемых бит равно 4, то тогда при декодировании эту битовую строку следует трактовать как 0000.

При кодировании типа BIT STRING может быть использован как примитивный метод кодирования, так и конструктивный. Конструктивный метод кодирования битовых строк позволяет просто логически разбить одну битовую строку на множество более мелких. При декодировании такой битовой строки, закодированной конструктивным методом, значения из всех вложенных битовых строк объединяются в одну большую битовую строку. В случае использования конструктивного метода задания битовых строк в блоке значения должны быть закодированы только битовые строки, причём у всех битовых подстрок, кроме последней, октет со значением не используемых бит должен быть равен нулю.

Приведем пример конструктивного кодирования битовых строк. Допустим начальная битовая строка содержит значения 0B 0B 0F, причем количество не используемых бит равно 4. Теперь разобьем эту строку на 3 вложенных битовых подстроки и получим следующую последовательность октетов, кодирующую начальную битовую строку в конструктивной форме:

23 0C

03 02 00 0B
03 02 00 0B
03 02 04 0F

Здесь в первых двух октетах приведены идентификационный октет для конструктивной формы кодирования битовой строки и общая длина значения конструктивной формы (12). Далее задано кодирование битовой строки со значением 0B (количество не используемых бит равно нулю!). Потом еще раз закодирована битовая строка со значением 0B (опять обязательно количество не используемых бит должно быть равно нулю). Последней закодирована битовая строка 0F, у которой задано количество не используемых бит равное 4. Таким образом, соединяя последовательно закодированные битовые подстроки, получим изначально закодированную битовую строку 0B 0B 0 (4 последних бита не используется, а следовательно F = 1111 отсекается от окончательного значения).

Тип OCTET STRING предназначен для хранения простых последовательностей байт (октетов). То есть в этот тип можно закодировать всё что угодно! В этом типе можно кодировать даже содержимое файлов операционной системы. Одно "но" в случае использования "неявного задания длины" (см. главу про основы кодирования в ASN.1) для типа OCTET STRING следует самостоятельно отслеживать появление в кодируемом потоке появления двух нулевых октетов подряд (00 00) и создавать дополнительные вложенные подстроки, разделяющие нулевые октеты во входном потоке октетов.

Глава 8. Кодирование префиксных типов

Зачастую необходимо различать закодированные в одном блоке значения элементы одного и того же типа, следующие один за другим (например, при кодировании типа SET, где порядок вложенных элементов может быть произвольным). Для этого применяют дополнительное кодирование элементов в блоках с новыми, специфичными тэгами. Например, типа REAL имеет класс тэга UNIVERSAL (00) и номер тэга 9. Для того чтобы логически отличить два подряд идущих значения REAL применяют дополнительное, "обёрточное" кодирование значения. Для этого используют классы тэгов кроме класса UNIVERSAL (00). То есть вводят новый тип, со своим отличительным номером тэга и классом тэга, отличным от UNIVERSAL (00), а в качестве значения для этого нового типа кодируется весь стандартный блок закодированного типа REAL (новый тип "инкапсулирует" всё закодированное значение другого типа, вместе с блоками идентификационного октета и блоками длины).

Например, "обернем" так закодированное значение для типа REAL для значения 0.15625 (кодированное в двоичной форме, основание равно 2, см. главу про кодирование REAL). Полностью это значение кодируется пятью октетами 09 03 80 FB 05. Для внешней "обертки" используем класс тэга PRIVATE (11) и номер тэга выберем равный 2. Так как в качестве значения для внутреннего блока будет использоваться не примитивный тип, а закодированное стандартное значение для типа REAL, то внешняя "обертка" будет иметь конструктивную форму кодирования. Следовательно, полностью тип REAL вместе с "оберткой" будет кодироваться с помощью октетов E2 05 09 03 80 FB 05.

В случае, когда между получателем закодированной информации и отправителем существует заранее согласованная и известная схема ASN.1 сообщений, то при использовании "оберточного кодирования" можно не указывать значение информационного октета для внутреннего, "обёрнутого", значения. В этом случае в качестве значения для "обёртки" будет уже использоваться примитивный тип и

закодированное значение можно записать как C2 03 80 FB 05 (здесь C2 - указание на применение класса тэга PRIVATE + применение примитивной формы кодирования + номер тэга равен 2; 03 - длина блока значения; оставшиеся октеты - блок значения, непосредственно взятый из блока значения для стандартного кодирования типа REAL). Таким образом можно сказать, что использование заранее согласованных схем ASN.1 позволяет кодировать в выбранных местах общего ASN.1 сообщения выбранные типы с помощью выбранных значений как классов тэгов, так и номеров классов тэгов (полная свобода действий!).

В заключение скажу несколько слов о нотации, которой обозначаются "префиксные типы". Рассмотрим несколько примеров:

Type1 ::= [0] BOOLEAN

Здесь описывается Type1, который имеет класс тэга "Context-specific" $(10)_2$, номер тэга равный 0 и конструктивную форму кодирования. В блоке значения для Type1 передается полностью кодированное значение стандартного типа BOOLEAN.

Type2 ::= [PRIVATE 2] IMPLICIT BOOLEAN

Здесь описывается тип Type2, который имеет класс тэга "Private" $(11)_2$, номер тэга равный 2 и примитивную форму кодирования. То есть в блоке значения для Type2 передается только соответствующий блок значения из стандартно закодированного типа BOOLEAN (идентификационный блок и блок длины теперь берутся из Type2).

То есть по умолчанию в ASN.1 в качестве класса тэга применяется "Context-specific" $(10)_2$, а также применяется конструктивная форма кодирования. То есть первый пример может быть записан в эквивалентном виде (хотя такая запись и не может непосредственно присутствовать в файле, описывающем типы ASN.1):

Type1 ::= [CONTEXT 0] EXPLICIT BOOLEAN

Кстати формально полный эквивалент стандартному типу, например BOOLEAN, может быть записан как:

BOOLEAN_Eq ::= [UNIVERSAL 1] IMPLICIT BOOLEAN

Еще одна интересная ситуация возникает в случае применения нотации IMPLICIT к изначально конструктивным типам, например к типу SEQUENCE. В этом случае новый тип будет также иметь конструктивную форму, а блок значения для нового типа всё так же будет браться из блока значения кодируемого типа SEQUENCE. Таким образом для нотации IMPLICIT можно обозначить следующие правила:

1. Блок значения для нового типа всегда берётся из блока значения кодируемого типа;
2. Использование конструктивного или примитивного кодирования зависит от кодируемого типа - тип кодирования нового типа эквивалентен типу кодирования, применяемому в кодируемом типе;

Например, в приведенной ниже нотации тип PrimType не смотря на применение IMPLICIT будет иметь конструктивный тип:

ConstrType ::= [0] REAL
PrimType ::= [PRIVATE 2] IMPLICIT ConstrType

В заключение скажу, что для типа CHOICE по правилам кодирования ASN.1 должна всегда применятся нотация EXPLICIT (конструктивное кодирование).

Общие правила:

- Нотация по-умолчанию: EXPLICIT;
- Нотация может быть различной для различных модулей схем ASN.1 и описывается в заголовке модуля с помощью DEFINITIONS EXPLICIT TAGS или DEFINITIONS IMPLICIT TAGS;
- Для нотации EXPLICIT всегда применяется конструктивное кодирование;
- Для нотации IMPLICIT фактически происходит только замена тэга на новое значение. Однако для типа CHOICE делается исключение: для данного типа всегда применяется конструктивное кодирование;

Глава 9. Кодирование типа SEQUENCE

Тип SEQUENCE служит для логической группировки (объединения) закодированных значений для различных типов. Фактически само название SEQUENCE (в переводе "последовательность") указывает на область применения этого типа. Порядок следования типов в последовательности заранее определен и не может быть изменен.
Кодирование для этого типа является "конструктивным", то есть в блоке с закодированным значением содержаться дополнительные подблоки, кодирующие отдельные значения. За более подробным описанием отсылаю читателя к главе 1 этой статьи.

Пример кодирования типа SEQUENCE. Предположим, что в последовательности должны быть закодированы два числа: одно целое (INTEGER), равное -128, и одно число с плавающей точкой (REAL) равное 0.15625 (представленное в разложении по основанию 2). Из соответствующих предыдущих глав можно узнать, что целое число кодируется как последовательность октетов (02 01 80), а число с плавающей запятой кодируется как последовательность октетов (09 03 80 FB 05). Тогда тип SEQUENCE, содержащий эти два закодированных числа будет кодироваться в виде:

30 08 02 01 80 09 03 80 FB 05

Здесь первый октет является идентификационным октетом и информирует о том, что закодировано значение типа SEQUENCE и что используется конструктивный метод кодирования. Второй октет хранит количество октетов, в которых закодировано значение типа SEQUENCE. Следующие три октета представляют закодированное значение целого числа (-128), а последние 5 октетов - значение закодированного числа с плавающей точкой (0.15625).

Глава 10. Кодирование типа SET

Кодирование типа SET фактически полностью соответствует кодированию для типа SEQUENCE за тем исключением, что порядок следования типов, закодированных с помощью SET, может изменяться. То есть если для примера взять числа из главы про тип SEQUENCE, то фактически мы получаем два возможных (абсолютно равноценных) варианта кодирования их для типа SET:

Вариант 1: 31 08 02 01 80 09 03 80 FB 05
Вариант 2: 31 08 09 03 80 FB 05 02 01 80

Второй вариант просто помещает закодированное целое число после закодированного числа с плавающей точкой.

В типе SET также могут быть закодированы два (и более) значений, имеющих одинаковые типы. При декодировании их значения различаются с помощью применения "префиксных типов". О них уже было рассказано в главе 8.

Глава 11. Кодирование типа BOOLEAN

Тип BOOLEAN может кодировать только два значение - или TRUE (истина), или FALSE (ложь). В случае если кодируется значение FALSE, то в блоке значения должен быть только один октет равный 00. В случае если кодируется значение TRUE то в блоке значения должен быть только один октет, значение которого отлично от нуля. То есть следующие два варианта кодирования TRUE для типа BOOLEAN эквивалентны:

Вариант 1: 01 01 01
Вариант 2: 01 01 FF

Глава 12. Кодирование типа NULL

Значение типа NULL всегда постоянно и всегда кодируется всего двумя октетами 05 00, где первый октет является информационным октетом, а второй октет - октет длины, который всегда кодирует нулевую длину.