Network Working Group M. Eisler, Ed.
Request for Comments: 4506 Network Appliance, Inc.
STD: 67 May 2006
Obsoletes: 1832
Category: Standards Track

XDR – стандарт внешнего представления данных

XDR: External Data Representation Standard

Статус документа

Этот документ содержит проект стандартного протокола для сообщества Internet и служит запросом для обсуждения в целях развития протокола. Информацию о текущем состоянии стандартизации протокола можно найти в документе Internet Official Protocol Standards (STD 1). Документ может распространяться свободно.

Тезисы

В этом описан стандартный протокол XDR[1] в его современном состоянии. Документ служит заменой RFC 1832.

Оглавление

[1] External Data Representation Standard — Стандарт внешнего представления данных.

1. Введение

XDR является стандартом для описания и представления данных. Он полезен при обмене данными между компьютерами с разной архитектурой и применяется при взаимодействии таких систем, как SUN WORKSTATION, VAX, IBM-PC, Cray. XDR работает на уровне представления ISO и является аналогом X.409, ISO ASN[1]. Основное различие между ними заключается в том, что XDR использует неявную типизацию, а X.409 - явную.

XDR использует язык описания формата данных. Этот язык служит лишь для этой цели и не может применяться для программирования. Этот язык обеспечивает компактное описание сложных форматов данных. Графическое представление форматов (неформальный язык) с ростом сложности формата становится непонятным. Язык XDR похож на язык программирования C [KERN], как Courier [COUR] похож на Mesa. Протоколы типа ONC RPC[2] и NFS[3] используются в XDR для описания формата данных.

Стандарт XDR использует допущение о переносимости байтов (или октетов), которые определяются, как 8 битов данных. Конкретное устройство кодирует байты в различных средах так, чтобы другие устройства могли их декодировать без потери информации. Например, стандарт Ethernet использует формат представления little-endian [COHE], когда младший бит указывается первым.

2. Отличия от RFC 1832

Этот документ не вносит технических изменений в RFC 1832 и публикуется с целью включения вопросов согласования с IANA, дополнительных вопросов безопасности и разделения нормативных и дополнительных ссылок.

3. Размер базового блока

Для представления всех объектов требуется блок из 4 байтов (или 32 битов) данных. Байты нумеруются от 0 до n-1. Байты считываются или записываются в некий байтовый поток так, что байт m всегда предшествует байту m+1. Если для хранения данных нужно n байтов и n не кратно 4, после n байтов данных будет размещено (r) (от 1 до 3 байтов[4]) с нулевыми значениями для того, чтобы общее число байтов было кратно 4.

В документе приводятся простые графические схемы для пояснения и сравнения. В большинстве иллюстраций каждый прямоугольник обозначает 1 байт. Скобки между прямоугольниками-байтами указывают на возможность присутствия (при необходимости) других байтов.

```
+--------+--------+...+--------+--------+...+--------+
| байт 0 | байт 1 |...|байт n-1|    0   |...|    0   |   BLOCK
+--------+--------+...+--------+--------+...+--------+
|<-----------n байтов--------->|<------r байтов----->|
|<------------n+r (где (n+r) mod 4 = 0)>------------>|
```

4. Типы данных XDR

В последующих параграфах описываются типы данных, определенные в стандарте XDR, приведено их формальное определение и графическое представление.

Для каждого типа данных в языке показана общая декларация парадигмы. Отметим, что угловые скобки (< и >) означают последовательность данных переменного размера, а квадратные скобки ([и]) указывают последовательность данных фиксированного размера. Буквы n, m и r обозначают целые числа (integer). Полная спецификация языка и более формальные определения терминов (таких, как «идентификатор» и «декларация») приведены в разделе 6. Спецификация языка XDR.

Для некоторых типов данных приведены более конкретные примеры. Расширенные примеры описания данных даны в разделе 7. Пример описания данных XDR.

4.1. Integer - целое число

Целое число со знаком в XDR представляет собой 32 бита данных, с помощью которых можно представить значения в диапазоне [-2147483648, 2147483647]. Целое число представляется в форме дополнения до двух. Старший байт имеет номер 0, младший — 3. Представление целых чисел показано ниже:

```
int identifier;

  (MSB)                   (LSB)
+-------+-------+-------+-------+
|байт 0 |байт 1 |байт 2 |байт 3 |                    INTEGER
+-------+-------+-------+-------+
<------------32 бита------------>
```

4.2. Unsigned Integer - целое число без знака

Целое число без знака в XDR представляет собой 32 бита, с помощью которых можно представить значения из диапазона [0, 4294967295]. Старший байт числа имеет номер 0, младший - 3. Представление показано ниже.

[1]Abstract Syntax Notation.
[2]Remote Procedure Call - удаленный вызов процедур.
[3]Network File System - сетевая файловая система.
[4]В оригинале сказано от 0 до 3, но 0 байтов добавляется при n кратном 4, а в начале предложения сказано «Если n не кратно 4». Особенности языка. *Прим. перев.*

```
unsigned int identifier;
  (MSB)                   (LSB)
+-------+-------+-------+-------+
|байт 0 |байт 1 |байт 2 |байт 3 |          UNSIGNED INTEGER
+-------+-------+-------+-------+
<------------32 бита------------>
```

4.3. Enumeration - перечисление

Перечисляемые элементы используют такое же представление, как целые числа со знаком. Перечисляемые элементы служат для описания подмножеств целых чисел. Перечисляемые данные декларируются следующим образом:

```
enum { name-identifier = constant, ... } identifier;
```

Например, три цвета red, yellow, blue можно описать, как перечисляемый тип:

```
enum { RED = 2, YELLOW = 3, BLUE = 5 } colors;
```

Будет ошибкой представление в качестве перечисляемого любого целого числа, не указанного при объявлении типа.

4.4. Boolean - логическое значение

Логические значения достаточно важны и часто используются, поэтому в стандарте для них выделен свой тип. Логические значения декларируются следующим образом:

```
bool identifier;
```

Это эквивалентно:

```
enum { FALSE = 0, TRUE = 1 } identifier;
```

4.5. Hyper Integer и Unsigned Hyper Integer

Стандарт также определяет 64-битовые (8 батов) целые числа, называемые hyper integer и unsigned hyper integer. Эти форматы являются простым расширением описанных выше форматов integer и unsigned integer. Числа представляются в форме дополнения до двух. Старший и младший байты имеют номера 0 и 7, соответственно.

```
hyper identifier; unsigned hyper identifier;

  (MSB)                                                   (LSB)
+-------+-------+-------+-------+-------+-------+-------+-------+
|байт 0 |байт 1 |байт 2 |байт 3 |байт 4 |байт 5 |байт 6 |байт 7 |
+-------+-------+-------+-------+-------+-------+-------+-------+
<----------------------------64 бита---------------------------->
                                          HYPER INTEGER
                                          UNSIGNED HYPER INTEGER
```

4.6. Floating-Point - число с плавающей запятой

Стандарт определяет тип данных с плавающей запятой - float (32 бита или 4 байта). Для этого типа используется представление IEEE для нормализованных чисел одинарной точности с плавающей запятой [IEEE]. Приведенные ниже три поля определяют число с плавающей запятой.

S — знак (0 для положительных чисел, 1 для отрицательных). 1 бит.

E — показатель степени (основание 2). 8 битов. Степени 0 соответствует значение 127 (Bias).

F — дробная часть мантиссы (основание 2). 23 бита.

Следовательно, число с плавающей запятой можно выразить формулой:

$$(-1)^{S} * 2^{(E-Bias)} * 1,F$$

Числа этого типа декларируются следующим образом:

```
float identifier;

+-------+-------+-------+-------+
|байт 0 |байт 1 |байт 2 |байт 3 |          SINGLE-PRECISION
S|   E   |           F           |       FLOATING-POINT NUMBER
+-------+-------+-------+-------+
1|<- 8 ->|<-------23 бита------>|
<------------32 бита------------>
```

Как старший и младший байты целого числа имеют номера 0 и 3, старший и младший биты числа одинарной точности с плавающей запятой имеют номера 0 и 31. Первый (старший) бит полей S, E и F имеет смещение 0, 1 и 9, соответственно. Отметим, что эти смещения указывают математические позиции битов, а **не** их реальное расположение, которое зависит от среды.

Представление 0 со знаком, бесконечности со знаком (переполнение) и денормализованных чисел (опустошение[1]) описаны в стандарте [IEEE]. В соответствии со спецификацией IEEE, значение NaN[2] зависит от системы и не должно интерпретироваться в XDR, как нечто, отличное от NaN.

4.7. Double-Precision Floating-Point (двойная точность)

Стандарт определяет представления чисел с плавающей запятой двойной точности - double (64 бита или 8 байтов). Представление использует стандарт IEEE для нормализованных чисел с плавающей запятой двойной точности [IEEE]. Для представления чисел используется три поля:

[1]Underflow.
[2]Not a number - не число.

S — знак (0 для положительных чисел, 1 для отрицательных). 1 бит.

E — показатель степени (основание 2). 11 битов. Степени 0 соответствует значение 1023 (Bias).

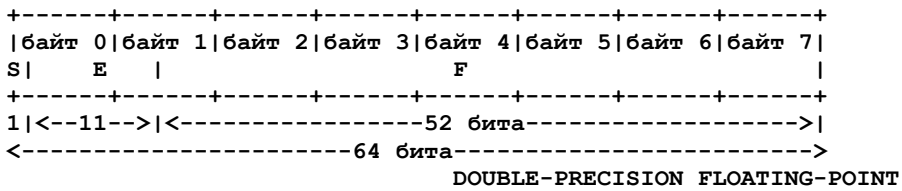
F — дробная часть мантиссы (основание 2). 52 бита.

Следовательно, число с плавающей запятой можно выразить формулой:

$$(-1)^S * 2^{(E-Bias)} * 1,F$$

Числа этого типа декларируются следующим образом:

```
double identifier;
```



DOUBLE-PRECISION FLOATING-POINT

Как старший и младший байты целого числа имеют номера 0 и 3, старший и младший биты числа двойной точности с плавающей запятой имеют номера 0 и 63. Первый (старший) бит полей S, E и F имеет смещение 0, 1 и 12, соответственно. Отметим, что эти смещения указывают математические позиции битов, а **не** их реальное расположение, которое зависит от среды.

Представление 0 со знаком, бесконечности со знаком (переполнение) и денормализованных чисел (опустошение[1]) описаны в стандарте [IEEE]. В соответствии со спецификацией IEEE, значение NaN[2] зависит от системы и не должно интерпретироваться в XDR, как нечто, отличное от NaN.

4.8. Quadruple-Precision Floating-Point (4-кратная точность)

Стандарт определяет представления чисел с плавающей запятой 4-кратной точности - quadruple (128 битов или 16 байтов). Представление чисел аналогично представлению чисел с плавающей запятой одинарной или двойной точности с использованием формата IEEE. Для представления чисел используется три поля:

S — знак (0 для положительных чисел, 1 для отрицательных). 1 бит.

E — показатель степени (основание 2). 15 битов. Степени 0 соответствует значение 16383 (Bias).

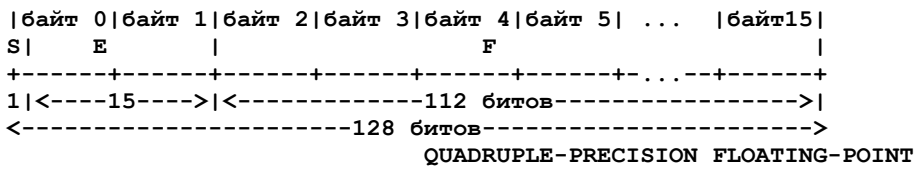
F — дробная часть мантиссы (основание 2). 112 битов.

Следовательно, число с плавающей запятой можно выразить формулой:

$$(-1)^S * 2^{(E-Bias)} * 1,F$$

Числа этого типа декларируются следующим образом:

```
quadruple identifier;
```



QUADRUPLE-PRECISION FLOATING-POINT

Как старший и младший байты целого числа имеют номера 0 и 3, старший и младший биты числа 4-кратной точности с плавающей запятой имеют номера 0 и 127. Первый (старший) бит полей S, E и F имеет смещение 0, 1 и 16, соответственно. Отметим, что эти смещения указывают математические позиции битов, а **не** их реальное расположение, которое зависит от среды.

Представление 0 со знаком, бесконечности со знаком (переполнение) и денормализованных чисел аналогичны числам с плавающий запятой одинарной и двойной точности [SPAR], [HPRE]. В соответствии со спецификацией IEEE, значение NaN для чисел 4-кратной точности зависит от системы и не должно интерпретироваться в XDR, как нечто, отличное от NaN.

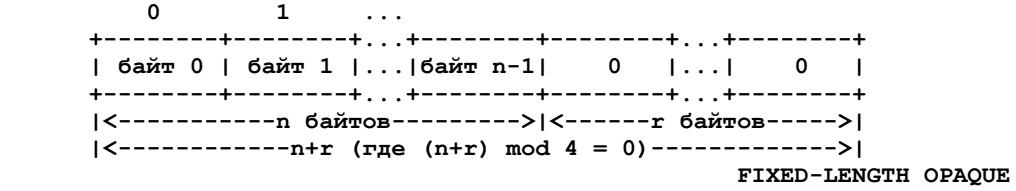
4.9. Неинтерпретируемые данные фиксированного размера

Иногда между машинами требуются передавать не интерпретируемые данные фиксированного размера. Такие «непрозрачные» (opaque) данные декларируются следующим образом:

```
opaque identifier[n];
```

где постоянная n задает (фиксированное) число байтов, которые должны содержаться в непрозрачных данных. Если n не кратно 4, после n следует r (от 1[3] до 3) байтов с нулевыми значениями, служащих для выравнивания размера до значения, кратного 4.

[1] Underflow.
[2] Not a number — не число.
[3] В оригинале сказано от 0, но в этом случае n будет кратно 4. *Прим. перев.*



4.10. Неинтерпретируемые данные переменного размера

Стандарт также поддерживает непрозрачные данные переменного (заданного) размера, определяемые, как последовательность из n (нумерация от 0 до n-1) произвольных байтов. Последовательность начинается с беззнакового целого числа n (см. выше), за которым следует n байтов данных.

Байт m в последовательности всегда предшествует байту m+1, а байт 0 всегда следует за полем размера (счетчик). Если значение n не кратно 4, к последовательности добавляется r (от 1[1] до 3) байтов со значением 0 для выравнивания размера по границе 4-байтового слова. Непрозрачные данные переменного размера декларируются следующим образом:

```
opaque identifier<m>;
```

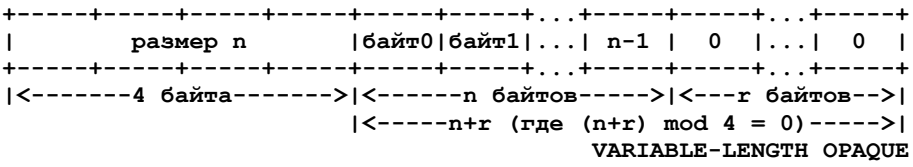
или

```
opaque identifier<>;
```

Постоянная m обозначает верхнюю границу числа байтов, которые могут содержаться в последовательности. Если m не задано (второй вариант декларации), предполагается максимальный размер 2^{32} - 1.

Постоянную m обычно можно найти в спецификации протокола. Например, протокол подачи может декларировать максимальный размер передаваемых данных 8192 байта следующим образом:

```
opaque filedata<8192>;
```



Превышение указанного в декларации максимального значения размера данных является ошибкой.

4.11. String - строка

Стандарт определяет строку из n (с номерами от 0 до n-1) символов ASCII, где размер строки задается целым числом без знака n (см. выше), а за ним следует n байтов самой строки. Байт m всегда предшествует в строке байту m+1, а байт 0 всегда следует за полем размера строки. Если значение n не кратно 4, после n байтов к строке добавляется r (от 1[1] до 3) байтов заполнения для выравнивания до размера, кратного 4. Строка байтов декларируется, как:

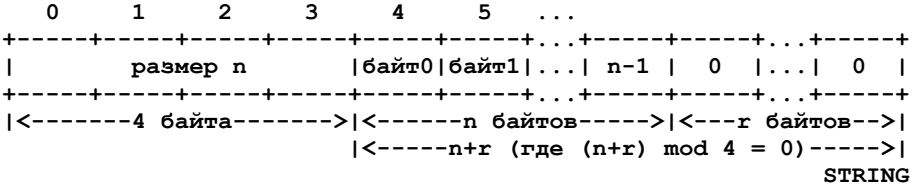
```
string object<m>;
```

или

```
string object<>;
```

Постоянная m обозначает верхнюю границу числа байтов, которые могут содержаться в последовательности. Если m не задано (второй вариант декларации), предполагается максимальный размер 2^{32} - 1. Постоянную m обычно можно найти в спецификации протокола. Например, протокол подачи может декларировать максимальный размер имени файла 255 байтов, как показано ниже:

```
string filename<255>;
```



Превышение указанного в декларации максимального значения размера данных является ошибкой.

4.12. Массив фиксированного размера

Декларация массива фиксированного размера из однородных элементов имеет вид:

```
type-name identifier[n];
```

Массивы фиксированного размера с элементами, имеющими номера от 0 до n-1 представляются путем кодирования (представления) отдельных элементов в порядке роста порядковых номеров от 0 до n-1. Размер каждого элемента в байтах кратен 4. Все элементы массива являются однотипными, но их размеры могут различаться. Например, в массиве строк все элементы имеют тип string, но могут различаться по размерам.

[1] В оригинале сказано от 0, но в этом случае n будет кратно 4. *Прим. перев.*

```
+---+---+---+---+---+---+---+---+...+---+---+---+---+
|   элемент 0   |   элемент 1   |...|  элемент n-1  |
+---+---+---+---+---+---+---+---+...+---+---+---+---+
|<--------------------n элементов------------------>|

                                          FIXED-LENGTH ARRAY
```

4.13. Массив переменного размера

Counted-массивы обеспечивают возможность создания массивов с переменным числом однотипных элементов. Массив представляется числом элементов n (целое число без знака), за которым следуют элементы массива с номерами от 0 до n-1. Декларация массива переменного размера имеет вид:

```
type-name identifier<m>;
```

или

```
type-name identifier<>;
```

Постоянная m задает максимально допустимое число элементов массива. Если значение m не задано (второй вариант) предполагается максимальное число элементов 2^{32} - 1.

```
  0  1  2  3
+--+--+--+--+--+--+--+--+--+--+--+...+--+--+--+--+
|     n     | элемент 0 | элемент 1 |...|элемент n-1|
+--+--+--+--+--+--+--+--+--+--+--+...+--+--+--+--+
|<-4 байта->|<-------------n элементов------------>|
                                                 COUNTED ARRAY
```

Если значение n превышает максимальное число элементов массива, возникает ошибка.

4.14. Structure - структура

Структуры декларируются следующим образом:

```
struct {
   component-declaration-A;
   component-declaration-B;
   ...
} identifier;
```

Компоненты структуры кодируются в порядке их объявления в декларации структуры. Размер каждого компонента кратен 4 байтам, но компоненты могут иметь разные размеры.

```
+-------------+-------------+...
| компонент A | компонент B |...                    STRUCTURE
+-------------+-------------+...
```

4.15. Discriminated Union - объединение с дискриминантом

Объединение этого типа состоит из дискриминанта, за которым следует тип, выбранный из множества предопределенных типов в соответствии со значением дискриминанта. Дискриминант может относиться к типу int, unsigned int или перечисляемому типу (например, bool). Типам компонент объединения (их называют arm - ветвь) предшествуют значения дискриминанта, определяющие их представление. Объединение с дискриминантом декларируется следующим образом:

```
union switch (discriminant-declaration) {
case discriminant-value-A:
   arm-declaration-A;
case discriminant-value-B:
   arm-declaration-B;
...
default: default-declaration;
} identifier;
```

За каждым ключевым словом case следует корректное значение дискриминанта. Принятая по умолчанию ветвь (arm) является необязательной. Если она не задана, корректное представление объединения не может принимать неуказанных значений дискриминанта. Размер принимаемой ветви всегда кратен 4 байтам.

Объединение с дискриминантом представляется дискриминантом, за которым следует представление принимаемой ветви.

```
  0   1   2   3
+---+---+---+---+---+---+---+---+
|  discriminant |  implied arm  |          DISCRIMINATED UNION
+---+---+---+---+---+---+---+---+
|<---4 байта--->|
```

4.16. Void - пустой тип

В XDR пустой (void) тип имеет размер 0 байтов. Пустой тип полезен для описания операций, которые не принимают данных на входе и/или не возвращают их на выходе. Этот тип также полезен в объединениях, где некоторые ветви включают данные, а другие не включают. Декларация пустого типа имеет вид:

```
void;
```

Пустой тип можно представить следующим образом:

```
          ++
          ||                                          VOID
          ++
        --><-- 0 байтов
```

4.17. Constant - константа

Декларация константы имеет вид:

```
const name-identifier = n;
```

Ключевое слово const служит для определения символьного имени константы, не декларируя каких-либо данных. Символьное имя может использоваться вместо константы в любом месте. Приведенный ниже пример определяет константу DOZEN со значением 12.

```
const DOZEN = 12;
```

4.18. Typedef - определение типа

Определение типа (typedef) не служит для декларирования каких-либо данных, но позволяет определять новые идентификаторы для декларирования данных. Синтаксис определения типа имеет вид:

```
typedef declaration;
```

Имя нового типа реально является именем переменной в декларативной части typedef. Например, можно определить новое имя типа eggbox на основе существующего типа egg:

```
typedef egg eggbox[DOZEN];
```

Переменные, декларируемые с использованием нового имени типа, имеют тот же тип, который указан для нового имени типа в typedef, если рассматривать его, как переменную. Например, две приведенных ниже декларации являются эквивалентными объявлениями переменной fresheggs:

```
eggbox  fresheggs; egg     fresheggs[DOZEN];
```

Для случаев, когда typedef включает определение struct, enum или union, существует другой (предпочтительный) синтаксис. В общем случае typedef вида:

```
typedef <<struct, union, or enum definition>> identifier;
```

можно преобразовать в другую форму, удалив часть typedef и поместив идентификатор после ключевого слова struct, union или enum вместо указания идентификатора в конце декларации. Ниже показаны два варианта определения типа bool:

```
typedef enum {     /* использование typedef */
   FALSE = 0,
   TRUE = 1
} bool;

enum bool {        /* предпочтительный вариант */
   FALSE = 0,
   TRUE = 1
};
```

Второй вариант является более предпочтительным, поскольку для определения нового имени типа не нужно просматривать декларацию до конца.

4.19. Optional-Data

Optional-data представляет собой один из вариантов объединения, который используется достаточно часто и по этой причине для него имеется специальный синтаксис декларирования. Декларация вида:

```
type-name *identifier;
```

эквивалентна декларации объединения:

```
union switch (bool opted) {
case TRUE:
   type-name element;
case FALSE:
   void;
} identifier;
```

Она также эквивалентна приведенной ниже декларации массива переменного размера, поскольку логическую переменную opted можно рассматривать, как размер массива:

```
type-name identifier<1>;
```

Тип Optional-data не интересен сам по себе, но он часто применяется для описания рекурсивных структур данных типа связанных списков или деревьев. Например, ниже определен тип stringlist, представляющий собой список (возможно пустой) строк произвольной длины:

```
struct stringentry {
   string item<>;
   stringentry *next;
};

typedef stringentry *stringlist;
```

Эту декларацию можно заменить эквивалентной декларацией объединения:

```
union stringlist switch (bool opted) {
case TRUE:
   struct {
      string item<>;
      stringlist next;
   } element;
case FALSE:
   void;
};
```

или массива переменного размера:

```
struct stringentry {
   string item<>;
   stringentry next<1>;
};

typedef stringentry stringlist<1>;
```

Оба эти варианта «скрывают» тип stringlist, поэтому явная декларация optional-data является более предпочтительной. Тип optional-data также коррелирует с представлением рекурсивных структур данных в языках программирования высокого уровня (типа Pascal или C) с помощью указателей. Фактически синтаксис совпадает с синтаксисом языка C для указателей.

4.20. Направления дальнейшего развития

В стандарте XDR нет представления битовых полей и битовых массивов, поскольку стандарт базируется на байтовом представлении. Отсутствуют также упакованные (с двоичным кодированием) десятичные дроби.

Назначением стандарта XDR не было описание всех типов данных, которые могут передаваться между машинами. Стандарт, скорее, описывает наиболее распространенные типы данных языков высокого уровня типа Pascal или C, чтобы написанные на этих языках приложения могли обмениваться данными через ту или иную среду.

Можно представить расширения XDR, которые позволят описать практически все существующие протоколы (такие, как TCP). Минимальным требованием для реализации этого является поддержка различных размеров блока и разного порядка байтов. Описанный здесь вариант XDR можно рассматривать, как 4-байтовый вариант big-endian более широкого семейства XDR.

5. Обсуждение

(1) Зачем нужен язык описания данных? Чем плохи диаграммы?

Существует множество преимуществ использования языков описания данных типа XDR по сравнению с использованием диаграмм. Языки являются более формальными, чем диаграммы и позволяют снизить число неоднозначностей в описаниях. Язык проще для понимания и позволяет меньше думать о мелких деталях битового представления данных. Кроме того, существуют близкие аналогии между типами XDR и типами данных в языках высокого уровня таких, как C или Pascal. Это существенно упрощает реализацию модулей кодирования и декодирования данных XDR. Наконец, спецификация языка, как таковая, представляет собой строку ASCII, которая может предаваться между машинами для интерпретации «на лету».

(2) Почему в XDR применяется только один порядок байтов?

Поддержка разных вариантов порядка байтов требует наличия протоколов верхнего уровня для определения используемого порядка байтов. Поскольку XDR не является протоколом, он не может реализовать такую функцию. Преимущество использования единственного порядка байтов заключается в том, что данные в формате XDR могут быть записаны, например, на магнитную ленту и любая машина сможет интерпретировать их, поскольку для определения порядка данных не требуется протокол верхнего уровня.

(3) Почему XDR использует порядок байтов big-endian, а не little-endian?

Разве это «справедливо» по отношению с машинам типа VAX(r), которые вынуждены преобразовывать порядок байтов?

Выбор любого из двух порядков будет «несправедливым» по отношению к части машин. Многие архитектуры, включая Motorola 68000 и IBM 370, поддерживают порядок байтов big-endian.

(4) Почему XDR использует 4-байтовые блоки?

Выбор размера блока XDR является компромиссом. Выбор меньшего размера (скажем, 2) позволил бы представлять более экономно, но вызвал бы проблемы на машинах, которые не выравнивают данные по такой границе. Выбор большего размера (скажем, 8) обеспечил бы соответствие с выравниванием данных практически на всех машинах, но привел бы к неоправданному росту заполнения. Поэтому было выбрано компромиссное значение. 4-байтовые блоки обеспечивают достаточно экономное представление данный и выравнивание по 4-байтовой границе поддерживается большинством машин за исключением такой «экзотики», как 8-байтовые Cray.

(5) Зачем данные переменного размера дополняются нулями?

Желательно обеспечить одинаковое представление одних и тех же данных на всех машинах, поскольку это упрощает сравнение данных и вычисление контрольных сумм. Заполнение нулями неиспользуемых байтов обеспечивает однотипность представления.

(6) Почему не используется явной типизации данных?

Издержки на типизацию данных превосходят незначительные преимущества, которые она может дать. Одной из сложностей, связанных с типизацией, является необходимость поддержки поля типа. Другие издержки связаны с интерпретацией поля типа и соответствующими операциями. Кроме того, большинство протоколов знает

ожидаемый тип данных, поэтому типизация будет избыточной информацией. Однако можно получить преимущества типизации данных и при использовании XDR. Одним из вариантов является представление строки, описывающей представляемые данные XDR, и самих данных. Другим вариантом является является присвоение значений для всех типов XDR и определение универсального типа, который служит дискриминантом значений, описывающих типы данных.

6. Спецификация языка XDR

6.1. Соглашения о нотации

В данной спецификации для описания языка XDR применяется расширенная нотация BNF[1]. Ниже приведено краткое описание нотации:

(1) Символы |, (,), [,], " и * имеют специальные значения.

(2) Терминальными символами являются строки любых символов, заключенные в двойные кавычки (").

(3) Нетерминальными символами являются строки символов, не имеющих специального значения.

(4) Варианты разделяются символом |.

(5) Необязательные элементы заключаются в квадратные скобки.

(6) Элементы группируются путем заключения их в скобки.

(7) Символ *, следующий за неким элементом, означает возможность вхождения любого числа таких элементов.

В качестве примера расмотрим запись:

```
"a " "very" (", " "very")* [" cold " "and "] " rainy " ("day" | "night")
```

которой соответствует бесконечное множество строк. Варианты таких строк включают:

```
"a very rainy day"
"a very, very rainy day"
"a very cold and rainy day"
"a very, very, very cold and rainy night"
```

6.2. Лексические замечания

(1) Комментарии начинаются символами /* и завершаются символами */.

(2) Пробельные символы служат в качестве разделителей элементов, а в остальном игнорируются.

(3) Идентификатор представляет собой букву, за которой могут следовать дополнительные буквы, цифры и символы подчеркивания (_). Регистр букв в идентификаторах не принимается во внимание.

(4) Десятичная константа выражает число по основанию 10 и представляет собой последовательность цифр. Первая цифра последовательности должна быть отлична от 0, последовательности может предшествовать знак минус (-).

(5) Шестнадцатеричная константа выражает число по основанию 16 и должна начинаться с последовательности 0x, за которой следует одна или множество шестнадцатеричных «цифр» (A, B, C, D, E, F, a, b, c, d, e, f, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

(6) Восьмеричная константа выражает число по основанию 8 и всегда начинается с символа 0, за которым следует одна или множество восьмеричных цифр (0, 1, 2, 3, 4, 5, 6, 7).

6.3. Синтаксическая информация

Декларация:

```
  type-specifier identifier
| type-specifier identifier "[" value "]"
| type-specifier identifier "<" [ value ] ">"
| "opaque" identifier "[" value "]"
| "opaque" identifier "<" [ value ] ">"
| "string" identifier "<" [ value ] ">"
| type-specifier "*" identifier
| "void"
```

Значение:

```
  constant
| identifier
```

Константа:

```
decimal-constant | hexadecimal-constant | octal-constant
```

[1]Back-Naur Form - форма Бэкуса-Наура.

Спецификатор типа:

```
          [ "unsigned" ] "int"
        | [ "unsigned" ] "hyper"
        | "float"
        | "double"
        | "quadruple"
        | "bool"
        | enum-type-spec
        | struct-type-spec
        | union-type-spec
        | identifier
```

Спецификация перечисляемого типа:

```
        "enum" enum-body
```

Перечисление:

```
        "{"
           ( identifier "=" value )
           ( "," identifier "=" value )*
        "}"
```

Спецификация структуры:

```
        "struct" struct-body
```

Структура:

```
        "{"
           ( declaration ";" )
           ( declaration ";" )*
        "}"
```

Спецификация объединения:

```
        "union" union-body
```

Объединение:

```
        "switch" "(" declaration ")" "{"
           case-spec
           case-spec *
           [ "default" ":" declaration ";" ]
        "}"
```

Спецификация вариантов:

```
       ( "case" value ":")
       ( "case" value ":") *
      declaration ";"
```

Определение постоянной:

```
        "const" identifier "=" constant ";"
```

Определение типа:

```
          "typedef" declaration ";"
        | "enum" identifier enum-body ";"
        | "struct" identifier struct-body ";"
        | "union" identifier union-body ";"
```

Определение:

```
          type-def
        | constant-def
```

Спецификация:

```
          definition *
```

6.4. Замечания по синтаксису

(1) Следующие слова являются ключевыми и не могут использоваться в качестве идентификаторов: bool, case, const, default, double, quadruple, enum, float, hyper, int, opaque, string, struct, switch, typedef, union, unsigned и void.

(2) В качестве значений размера массивов могут использоваться только числа без знака. При использовании идентификатора в качестве размера этот идентификатор должен быть объявлен, как константа без знака с помощью определения const.

(3) Идентификаторы констант и типов в рамках данной спецификации используют общее пространство имен и должны быть уникальными в зоне их действия.

(4) Аналогично имена переменных должны быть уникальными в рамках структур и объединений. Вложенные структуры и объединения создают новые области действия переменных.

(5) Дискриминант объединения должен иметь тип, приводимый к целому числу. Т. е. он может относиться к типам int, unsigned int, bool, перечисляемым или typedef. Такое же требованием применяется к вариантам. Кроме того, каждое значение варианта может использоваться в декларации объединения не более одного раза.

7. Пример описания данных XDR

Ниже приведено краткое описание данных XDR для файла (file), который может передаваться с одной машины на другую.

```
const MAXUSERNAME = 32;     /* максимальный размер имени пользователя */
const MAXFILELEN = 65535;   /* максимальный размер файла              */
const MAXNAMELEN = 255;     /* максимальный размер имени файла        */

/*
 * Типы файлов:
 */
enum filekind {
   TEXT = 0,       /* ascii-данные */
   DATA = 1,       /* raw-данные   */
   EXEC = 2        /* исполняемый  */
};

/*
 * Информация о файлах по их типам
 */
union filetype switch (filekind kind) {
case TEXT:
   void;                           /* нет дополнительной информации */
case DATA:
   string creator<MAXNAMELEN>;     /* создатель данных              */
case EXEC:
   string interpretor<MAXNAMELEN>; /* программный интерпретатор     */
};

/*
 * Полный файл:
 */
struct file {
   string filename<MAXNAMELEN>; /* имя файла          */
   filetype type;               /* информация о файле */
   string owner<MAXUSERNAME>;   /* владелец файла     */
   opaque data<MAXFILELEN>;     /* дата файла         */
};
```

Предположим, что пользователь с именем john хочет сохранить свою программу sillyprog на языке lisp, содержащую просто строку данных (quit). Ниже показано представление файла:

Смещение	HEX-байты	ASCII	Комментарии
0	00 00 00 09	....	Размер имени файла (9)
4	73 69 6c 6c	sill	Символы имени файла
8	79 70 72 6f	ypro	… дополнительные символы ...
12	67 00 00 00	g...	… и 3 байта заполнения (0)
16	00 00 00 02	....	Тип файла (EXEC = 2)
20	00 00 00 04	....	Размер имени интерпретатора (4)
24	6c 69 73 70	lisp	Имя интерпретатора
28	00 00 00 04	....	Размер имени владельца (4)
32	6a 6f 68 6e	john	Имя владельца
36	00 00 00 06	....	Число байтов данных в файле
40	28 71 75 69	(qui	Байты данных файла ...
44	74 29 00 00	t)..	… и 2 байта заполнения (0)

8. Вопросы безопасности

XDR является языком описания данных, а не протоколом и, следовательно, не связан напрямую с вопросами безопасности. Протоколы, передающие данные в формате XDR (например, NFSv4), сами отвечают за вопросы безопасности при передаче данных.

Следует аккуратно кодировать и декодировать данные. Известные риски, которых можно избежать, включают:

- Атаки на переполнение буфера. По возможности протоколы следует определять с явными пределами (путем использования нотации <[значение]> вместо < >) для элементов с типами данных переменного размера. Независимо от возможности явного ограничения переменных размеров элемента данного протокола, декодеры должны проверять размеры данных на входе в части предотвращения выхода за размеры приемных буферов.

- Нулевые октеты в значении типа string. Если естественным для декодера форматом являются строки, завершающиеся nul-символом, видимый размер декодированного объекта будет меньше объема памяти, выделенного для строки. Некоторые интерфейсы возвращения (deallocation) памяти поддерживают параметр

размера. При обращении к такому интерфейсу вызывающая сторона будет скорей всего передавать размер строки до первого nul-символа, добавляя к нему 1. Это несоответствие может приводить к «утечке» памяти (поскольку возвращаться памяти будет меньше, чем было выделено), которая может приводить к системным отказам и атакам с целью отказа в обслуживании.

- Декодирование корректных символов ASCII, которые являются недопустимыми с точки зрения приложения. Например, некоторые операционные системы трактуют символ /, как разделитель компонент имени пути к файлу. Для протоколов, которые кодируют строки аргументов файловых операций, декодер должен обеспечить отсутствие символов / внутри компонент имени. В противном случае может быть создан файл с недопустимым символом / в имени, который будет сложено удалить и это, по сути, может служить атакой с целью отказа в обслуживании.

- Отказы в обслуживании, вызываемые рекурсивными подпрограммами кодирования и декодирования. Рекурсивное кодирование и декодирование может обрабатывать структурированные данные, которые сами по себе могут включать прямые или непрямые ссылки на структурированные данные (например, связные списки):

```
struct m {
  int x;
  struct m *next;
};
```

Программа кодирования или декодирования может рекурсивно вызывать сама себя каждый арз при обнаружении другого элемента struct m. Атакующий может создать длинный связный список элементов struct m в запросе или отклике и такой список может привести к переполнению стека для кодера или декодера. Программы кодирования-декодирования следует создавать без использования рекурсии или ограничивать длину списков[1].

9. Согласование с IANA

В будущем возможно (и весьма вероятно) добавление новых типов данных в XDR. Процесс добавления новых типов осуществляется в рамках стандартизации RFC и не включает регистрацию новых типов в IANA. Проекты стандартов RFC, которые обновляются или заменяются данным документов, должны быть отмечены соответствующим образом в базе данных RFC.

10. Торговые знаки и их владельцы

SUN WORKSTATION	Sun Microsystems, Inc.
VAX	Hewlett-Packard Company
IBM-PC	International Business Machines Corporation
Cray	Cray Inc.
NFS	Sun Microsystems, Inc.
Ethernet	Xerox Corporation.
Motorola 68000	Motorola, Inc.
IBM 370	International Business Machines Corporation

11. Стандарт ANSI/IEEE 754-1985

Определения NaN, нулей со знаком, бесконечности и денормализованных чисел для удобства заимствованы из [IEEE]. Определения чисел с плавающей запятой 4-кратной точности аналогичны определениям чисел обычной и двойной точности в [IEEE].

Ниже S обозначает бит знака, E - показатель степени, а F - дробную часть числа. Символ u указывает бит с неопределенным значения (0 или 1).

Для чисел с плавающей запятой:

Тип	S (1бит)	E (8 битов)	F (23 бита)
signalling NaN	u	255 (макс) (не менее 1 бита)	.0uuuuu---u
quiet NaN	u	255 (макс)	.1uuuuu---u
Отрицательная бесконечность	1	255 (макс)	.000000---0
Положительная бесконечность	0	255 (макс)	.000000---0
Отрицательный 0	1	0	.000000---0
Положительный 0	0	0	.000000---0

[1] Глубину рекурсии. *Прим. перев.*

Для чисел с плавающей запятой двойной точности:

Тип	S (1бит)	E (11 битов)	F (52 бита)
signalling NaN	u	2047 (макс) (не менее 1 бита)	.0uuuuu---u
quiet NaN	u	2047 (макс)	.1uuuuu---u
Отрицательная бесконечность	1	2047 (макс)	.000000---0
Положительная бесконечность	0	2047 (макс)	.000000---0
Отрицательный 0	1	0	.000000---0
Положительный 0	0	0	.000000---0

Для чисел с плавающей запятой 4-кратной точности:

Тип	S (1бит)	E (15 битов)	F (112 битов)
signalling NaN	u	32767 (макс) (не менее 1 бита)	.0uuuuu---u
quiet NaN	u	32767 (макс)	.1uuuuu---u
Отрицательная бесконечность	1	32767 (макс)	.000000---0
Положительная бесконечность	0	32767 (макс)	.000000---0
Отрицательный 0	1	0	.000000---0
Положительный 0	0	0	.000000---0

Субнормальные числа представляются следующим образом:

Точность	Показатель	Значение
Одинарная	0	$(-1)^S * 2^{-126} * 0,F$
Двойная	0	$(-1)^S * 2^{-1022} * 0,F$
Четырехкратная	0	$(-1)^S * 2^{-16382} * 0,F$

12. Нормативные документы

[IEEE] "IEEE Standard for Binary Floating-Point Arithmetic", ANSI/IEEE Standard 754-1985, Institute of Electrical and Electronics Engineers, August 1985.

13. Дополнительная литература

[KERN] Brian W. Kernighan & Dennis M. Ritchie, "The C Programming Language", Bell Laboratories, Murray Hill, New Jersey, 1978.

[COHE] Danny Cohen, "On Holy Wars and a Plea for Peace", IEEE Computer, October 1981.

[COUR] "Courier: The Remote Procedure Call Protocol", XEROX Corporation, XSIS 038112, December 1981.

[SPAR] "The SPARC Architecture Manual: Version 8", Prentice Hall, ISBN 0-13-825001-4.

[HPRE] "HP Precision Architecture Handbook", June 1987, 5954-9906.

14. Благодарности

Bob Lyon представлял Sun в рабочей группе ONC RPC в 1980-х. Компания Sun Microsystems, Inc. значится разработчиком RFC 1014. Raj Srinivasan и другие участники рабочей группы ONC RPC редактировали RFC 1014 в RFC 1832, который послужил основой для данного документа. Mike Eisler и Bill Janssen представили обзоры по реализациям данного стандарта. Kevin Coffman, Benny Halevy и Jon Peterson рассмотрели документ и дали свои замечания. Peter Astrand и Bryan Olson отметили несколько ошибок в RFC 1832, которые были исправлены в этом документе.

Адрес редактора

Mike Eisler

5765 Chase Point Circle

Colorado Springs, CO 80919

USA

Phone: 719-599-9026

EMail: email2mre-rfc4506@yahoo.com

Отправляйте комментарии по адресу: nfsv4@ietf.org

Перевод на русский язык

Николай Малых

nmalykh@protocols.ru

Подтверждение

Финансирование функций RFC Editor обеспечено IETF Administrative Support Activity (IASA).