Internet Engineering Task Force (IETF)
Request for Comments: 8949
STD: 94
Obsoletes: 7049
Category: Standards Track
ISSN: 2070-1721

C. Bormann
Universität Bremen TZI
P. Hoffman
ICANN

December 2020

Concise Binary Object Representation (CBOR)

Краткое представление двоичных объектов

Аннотация

Краткое представление двоичных объектов (Concise Binary Object Representation или CBOR) - это формат данных, разработанный специально для обеспечения очень малого размера кода и расширяемость без необходимости согласовывать версии. Эти цели отличают CBOR от предшествующих вариантов двоичной сериализации, таких как ASN.1 и MessagePack.

Этот документ отменяет RFC 7049, внося редакторские правки, новые детали и исправление ошибок при сохранении полной совместимости с форматом обмена RFC 7049. Документ не задаёт новой версии формата.

Статус документа

Документ относится к категории Internet Standards Track.

Документ является результатом работы IETF[1] и представляет согласованный взгляд сообщества IETF. Документ прошёл открытое обсуждение и был одобрен для публикации IESG[2]. Дополнительную информацию о стандартах Internet можно найти в разделе 2 в RFC 7841.

Информация о текущем статусе документа, найденных ошибках и способах обратной связи доступна по ссылке https://www.rfc-editor.org/info/rfc8949.

Оглавление

[1]Internet Engineering Task Force - комиссия по решению инженерных задач Internet.
[2]Internet Engineering Steering Group - комиссия по инженерным разработкам Internet.

1. Введение

Имеются сотни стандартизованных форматов для двоичного представления структурированных данных (binary serialization format). Некоторые из них предназначены для конкретных сфер информации, другие - для произвольных данных. В IETF наиболее известными форматами второй категории являются ASN.1 BER и DER [ASN.1].

Описываемый здесь формат следует некоторым целям, не вполне совпадающим с целями имеющихся форматов. Базовой моделью данных служит расширяемая версия модели JSON [RFC8259]. Важно отметить, что это не предложение по расширению грамматики RFC 8259 в целом, поскольку формат может быть не совместим с имеющимися вариантами JSON. В документе просто определяется своя модель данных, начинающаяся с JSON.

В Приложении E приведены некоторые имеющиеся двоичные форматы и обсуждается их соответствие целям CBOR.

Этот документ отменяет [RFC7049], внося редакторские правки, новые детали и исправление ошибок при сохранении полной совместимости с форматом обмена RFC 7049. Документ не задаёт новой версии формата.

1.1. Цели

Цели CBOR примерно в порядке снижения важности указаны ниже.

1. Представление должно однозначно кодировать наиболее общие типы данных, применяемые в стандартах Internet.

   - Должен предоставляться разумный набор базовых типов и структур данных с применением двоичного кодирования. Разумность здесь во многом определяется возможностями JSON с добавлением двоичных строк данных. К поддерживаемым структурам относятся массивы и деревья, а циклы и графы в виде решёток (lattice-style) не поддерживаются.

   - Не требуется однозначное кодирование всех типов данных, т. е. возможно представление, например, числа 7, разными способами.

2. Код для кодера или декодера должен быть компактным, чтобы поддерживать системы с ограниченной памятью, мощностью процессора и набора инструкций.
   - Кодер и декодер должны быть реализуемы очень небольшим объёмом кода (например, в ограниченных узлах класса 1, как определено в [RFC7228]).
   - Формату следует использовать современное машинное представление данных (например, без преобразования между двоичными и десятичными значениями).
3. Данные должны быть декодируемыми без описания схемы.
   - Как в JSON, кодирование данных должно быть самоописывающим, чтобы можно было создать универсальный декодер.
4. Сериализация должна быть разумно компактной, но компактность кода важней компактности данных.
   - Разумность здесь ограничивается JSON в качестве верхней границы размера и сложностью реализации, которая ограничивает усилия, затрачиваемые на обеспечение компактности. Применение общих схем сжатия или избыточного перебора битов нарушает этот подход.
5. Формат должен быть применим как для узлов с ограничениями, так и для приложений с большим объёмом данных.
   - Это значит, что он должен быть достаточно экономным в части ресурсов CPU, применяемых для кодирования и декодирования. Это актуально как для узлов с ограничениями, так и для возможного использования в приложениях с большими объёмами данных.
6. Формат должен поддерживать все типы данных JSON для преобразования в JSON и обратно.
   - Должен поддерживаться разумный уровень преобразования для данных, представляемых в пределах возможностей JSON. Должна быть возможность определить одностороннее преобразование в JSON для всех типов данных.
7. Формат должен быть расширяемым и для расширенных данных должно поддерживаться декодирование более ранними декодерами.
   - Формат рассчитан на использование в течение десятилетий.
   - Формат должен поддерживать расширяемость, допускающую откат, чтобы декодер, не поддерживающий расширения, мог декодировать сообщение.
   - Формат должен обеспечивать возможность расширения в будущих стандартах IETF.

1.2. Терминология

Ключевые слова **необходимо** (MUST), **недопустимо** (MUST NOT), **требуется** (REQUIRED), **нужно** (SHALL), **не нужно** (SHALL NOT), **следует** (SHOULD), **не следует** (SHOULD NOT), **рекомендуется** (RECOMMENDED), **не рекомендуется** (NOT RECOMMENDED), **возможно** (MAY), **необязательно** (OPTIONAL) в данном документе должны интерпретироваться в соответствии с BCP 14 [RFC2119] [RFC8174] тогда и только тогда, когда они выделены шрифтом, как показано здесь.

Термин «байт» применяется в обычном смысле как синоним октета (битов). Все многобайтовые значения представляются в сетевом порядке байтов (сначала старший байт, big-endian). Используемые в спецификации термины определены ниже.

Data item - элемент данных
: Единая часть данных CBOR. Структура элемента данных может включать вложенные элементы. Термин применяется как для элементов данных в формате представления, так и для абстрактной идеи, которая может быть выведена из него с помощью декодера. Первый может называться более конкретно кодированным элементом данных.

Decoder - декодер
: Процесс, декодирующий корректно сформированный элемент данных CBOR и делающий его доступным для приложения. Формально декодер состоит из синтаксического анализатора для разбивки ввода по синтаксическим правилам CBOR, а также семантического процессора для подготовки данных в пригодной для приложения форме.

Encoder - кодер
: Процесс, создающий (корректно сформированное) представление элемента данных CBOR из данных приложения.

Data Stream - поток данных
: Последовательность (возможно пустая) элементов данных, не связанных в более крупный элемент данных (см. [RFC8742]). Независимые элементы данных, составляющие поток иногда называют элементами данных верхнего уровня (top-level data item).

Well-formed - корректно сформированный
: Элемент данных, соответствующий синтаксической структуре CBOR. Корректно сформированный элемент данных использует начальные байты и строки байтов и/или элементы данных, предполагающие значения в соответствии с CBOR, и не включают после них посторонних данных. Декодеры CBOR по определению возвращают содержимое лишь корректно сформированных элементов данных.

Valid - действительный, пригодный
: Корректно сформированный элемент данных, следующий семантическим ограничениям CBOR (параграф 5.3).

Expected - ожидаемый
: Помимо обычного значения «ожидаемый» этот термин служит для описания требований сверх пригодности CBOR, которые приложение предъявляет к входным данным. Корректность формирования (полностью обрабатываемый), пригодность (проверка на пригодность базовым декодером) и ожидаемость (проверка приложением) формируют иерархию уровней пригодности.

Stream decoder - потоковый декодер

Процесс, декодирующий поток данных и делающий каждый из элементов данных в последовательности доступным для приложения по мере получения.

Термины и концепции для значений с плавающей точкой, таких как бесконечность (Infinity), не число (NaN - not a number), отрицательный 0 (negative zero), и субнормальное значение (subnormal), определены в [IEEE754].

При описании битовой арифметики и типов данных в документе применяется синтаксис, похожий на используемый в языке C [C], однако «..» означает диапазон с включением обеих границ, а надстрочный индекс - возвадение в степень (например, 2 в степени 64 обозначается как 2^(64)). В текстовой версии спецификации надстрочные индексы недоступны, поэтому применяется суррогатная нотация. Такая нотация не совсем подходит для данного RFC, поскольку её можно спутать с обозначением исключающего ИЛИ в языке C (здесь встречается лишь в приложениях, где нет примеров возведения в степень) и от читателя требуется осмотрительность при работе с текстовой версией[1].

В примерах и псевдокоде предполагается, что целые числа со знаком представлены дополнением до 2 и сдвиг числа со знаком вправо учитывает знак. Такие же допущения заданы в параграфах 6.8.1 (basic.fundamental) и 7.6.7 (expr.shift) версии C++ 2020 (доступна в виде финального черновика [Cplusplus20]).

Подобно 0x для обозначения шестнадцатеричных чисел, для двоичных чисел применяется префикс 0b. Для удобочитаемости в числа могут включаться символы подчёркивания, например, 0b00100001 (0x21) можно записать как 0b001_00001, чтобы подчеркнуть желательную интерпретацию битов в байте (здесь значение делится на части с 3 и 5 битами). Элементы данных в кодировке CBOR всегда указываются в нотации 0x или 0b и сначала интерпретируются как числа, подобно C, затем - как строки байтов с сетевым порядком, включая нулевые байты в начале.

Слова могут выделяться *курсивом* и в текстовом варианте документа это указывается символами подчеркивания (_italicized_) по обе стороны слова. Дословный текст (например, имена из языка программирования) может указываться `фиксированным` шрифтом и в текстовой версии документда для этого (не вполне однозначно) применяются двойные кавычки ("monospace"), которые применяются и в обычном своём значении.

2. Модели данных CBOR

CBOR явно указывает свою базовую модель данных, определяющую набор всех элементов данных, которые могут быть представлены в CBOR. Базовая модель данных может расширяться путём регистрации «простых значений» и тегов. Приложения могут затем создавать подмножество расширенной базовой модели данных для своих моделей.

В средах, где элементы данных могут быть представлены в базовой модели данных, можно реализовать базовые кодеры и декодеры CBOR (это обычно включает определение дополнительных типов данных реализации для элементов, не имеющих естественного представления в среде). Возможность обеспечить базовые кодеры и декодеры является явной целью разработки CBOR, однако многие приложения будут предоставлять свои кодеры и декодеры.

В базовой (не расширенной) модели данных, описанной в разделе 3, элементами данных могут быть:

- целые числа от -2^{64} до $2^{64}-1$, включительно;
- простые значения, указываемые числом от 0 до 255, но не являющиеся числами;
- значения с плавающей точкой, отличающиеся от целых чисел вне набора IEEE 754 binary64 (включая неконечные) [IEEE754];
- последовательность (возможно пустая) байтов (строка байтов);
- последовательность (возможно пустая) кодов Unicode (строка текста);
- последовательность (возможно пустая) элементов данных (массив);
- отображение (математическая функция) набора (возможно пустого) элементов данных (ключ) на элемент данных (значение), (отображение)
- помеченный тегом элемент данных, содержащий номер (целое число от 0 до $2^{64}-1$) и содержимое тега (элемент данных).

Отметим, что целые и действительные (floating-point) числа в этой модели различаются, даже когда значения чисел совпадают.

Отметим также, что варианты сериализации не видны на уровне базовой модели данных. Это преднамеренное отсутствие видимости включает число байтов кодированных значений с плавающей точкой. Это относится также к выбору кодирования для «аргумента» (раздел 3), такого как кодирование целого числа, кодирование размера строки текста или байтов, кодирование числа элементов в массиве, пар в отображении или номеров тегов.

2.1. Расширенные базовые модели данных

Базовая модель данных расширена в этом документе путём регистрации ряда простых значений и номеров тегов:

- false, true, null, undefined (простые значения, указываемые числами от 20 до 23, параграф 3.3);
- целые и действительные (floating-point) числа с большим диапазоном и точностью, чем указано выше (теги 2 - 5, параграф 3.4);
- типы данных приложения, такие как момент времени или даты и времени, определённые в RFC 3339 (теги 1 и 0, параграф 3.4)

Дополнительные элементы расширенной базовой модели данных могут быть (и были) определены через реестры IANA, созданные для CBOR. Даже при неизвестности такого расширения базовому кодеру или декодеру, использующие это расширение элементы данных могут передаваться в приложение или из него путём их

[1] В переводе применяется привычное обозначение, например 2^{64}. *Прим. перев.*

представления на интерфейсе приложения в рамках обычной базовой модели данных, т. е. как базовые простые значения или базовые теги.

Иными словами, базовая модель данных стабильна, как определено в этом документе, а расширенные базовые модели создаются путём регистрации новых простых значений или номеров тегов без сокращения базовой модели.

Имеются серьёзные основания ждать, что базовые кодеры и декодеры смогут представлять false, true, null (undefined намерено не указано) в форме, подходящей для среды программирования, а поддержка расширений модели данных по тегам является полностью необязательной и зависит от качества реализации.

2.2. Конкретные модели данных

Конкретная модель данных для протокола на основе CBOR обычно принимает подмножество расширенной базовой модели данных и назначает семантику приложения для элементов данных в этом подмножестве и его компонентах. При документировании таких моделей данных и задании типов элементов данных предпочтительно указывать типы их именами в базовой модели (negative integer, array), а не ссылаться на аспекты их представления CBOR (major type 1, major type 4).

Конкретные модели данных могут также задавать эквивалентность значений (включая разнотипные) для ключей отображений и свободы кодеров. Например, в базовой модели данных действительное отображение **может** иметь в качестве ключей 0 и 0.0, а кодеру **недопустимо** представлять 0.0 как целое число (major type 0, параграф 3.1). Однако, если конкретная модель данных заявляет эквивалентность целых и действительных значений одного числа, использование ключей 0 и 0.0 в одном отображении будет считаться дублированием, даже если они представляются разными базовыми типами, а это не допускается и кодер может представлять целочисленные действительные значения целым числом или наоборот, возможно для экономии размера закодированного значения.

3. Спецификация кодирования CBOR

Элемент данных CBOR (раздел 2) кодируется или декодируется через строку байтов, содержащую корректно сформированный, кодированный элемент данных, как описано в этом параграфе. Сводка кодирования приведена в таблице 7 Приложения B, индексом служит начальный байт. Кодер **должен** создавать только корректно сформированные элементы данных. Декодеру **недопустимо** возвращать элемент данных, если он получил на входе некорректно сформированный элемент данных CBOR (это не умаляет полезности средств диагностики и восстановления, которые могут сделать доступной некоторую информацию из повреждённого элемента данных CBOR).

Начальный байт каждого кодированного элемента данных содержит сведения об основном типе (major type - 3 старших бита, описанных в параграфе 3.1) и дополнительное значение (5 младших битов). С некоторыми исключениями, значение этих 5 байтов описывает способ загрузки целочисленного (без знака) аргумента, как описано ниже.

Меньше 24
: Значением аргумента является дополнительное значение.

24, 25, 26, 27
: Значение аргумента размещено в следующих 1, 2, 4 или 8 байтах (соответственно) с сетевым порядком. Для базового типа 7 и дополнительных значений 25, 26, 27 эти байты содержат не целочисленный аргумент, а число с плавающей точкой (см. параграф s3.3).

28, 29, 30
: Эти значения зарезервированы для будущих расширений CBOR. В текущей версии закодированный элемент будет некорректным по форме.

31
: Значение аргумента не выводится. Для базовых типов 0, 1, 6 закодированный элемент некорректен по форме. Для базовых типов 2 - 5 размер элемента не определён, а для типа 7 байт не указывает элемент данных, а просто завершает элемент неопределённого размера (см. параграф 3.2).

Начальный байт и любые дополнительные байты, служащие для построения аргумента, называют *головой* (_head_) элемента данных.

Назначение аргумента зависит от базового типа (major type). Например, для типа 0 аргументом служит значение самого элемента данных, а для типа 1 значение элемента данных вычисляется из аргумента, для типов 2 и 3 аргумент указывает размер следующей за ним строки данных в байтах, для типов 4 и 5 аргумент служит для определения числа вложенных элементов данных.

Если закодированная последовательность байтов завершается раньше конца элемента данных, это говорит о некорректной форме элемента данных. Если в закодированной последовательности байтов остаются байты после декодирования внешнего кодированного элемента, этот код не представляет один корректно сформированный элемент CBOR. В зависимости от приложения декодер может счесть кодирование некорректным или просто идентифицировать начало оставшихся байтов для приложения.

Реализация декодера CBOR может быть основана на таблице переходов со всеми 256 определёнными значениями начального байта (Таблица 7). В реализации с ограничениями декодер может применять структуру начального байта и следующих байтов для более компактного кода (в Приложении C показано, как это можно сделать).

3.1. Базовые типы

Ниже описаны базовые типы, дополнительные значения и другие байты, связанные с типом.

Базовый тип 0
: Целое число без знака от 0 до 2^{64}-1, включительно. Значением закоированного элемента является сам аргумент. Например, целое число 10 обозначается как байт 0b000_01010 (базовый тип 0, дополнительное значение 10). Целое число 500 будет иметь вид 0b000_11001 (базовый тип 0, дополнительное значение 25) с двумя байтами 0x01f4, указывающими десятичное значение 500.

Базовый тип 1

Отрицательное целое число от -2^{64} до -1, включительно. Значением элемента будет -1 за вычетом аргумента. Например, целое число -500 будет иметь вид 0b001_11001 (базовый тип 1, дополнительное значение 25) с двумя байтами 0x01f3, указывающими десятичное число 499.

Базовый тип 2

Строка байтов, число которых указывает аргумент. Например, строка из 5 байтов имеет начальный байт 0b010_00101 (базовый тип 2, дополнительное значение 5 для размера) с 5 байтами двоичного содержимого. Строка из 500 байтов будет иметь начальный байт 0b010_11001 (базовый тип 2, дополнительное значение 25 для указания 2-байтового поля размера), 2-байтовое поле 0x01f4, указывающее размер 500 и пятьсот байтов двоичного содержимого.

Базовый тип 3

Текстовая строка (раздел 2) в кодировке UTF-8 [RFC3629]. Число байтов в строке указывает аргумент. Строка с непригодной последовательностью UTF-8 считается корректно сформированной, но недействительной (параграф 1.2). Этот тип предназначен для систем, которым нужно интерпретировать или выводить человекочитаемый текст и позволяет отделить неструктурированные байты от текста, имеющего конкретное значение (в Unicode) и кодировку (UTF-8). В отличие от таких форматов, как JSON, символы Unicode в этом типе никогда не экранируются (escape). Таким образом, символ новой строки (U+000A) всегда представляется в строке как 0x0a, а не байтами 0x5c6e (символы \ и n) или 0x5c7530303061 (символы \, u, 0, 0, 0, a).

Базовый тип 4

Массив элементов данных. В других форматах массивы называют также списками, последовательностями и кортежами (последовательность CBOR немного отличается, [RFC8742]). Аргумент указывает число элементов данных в массиве, которые не обязаны быть однотипными. Например, массив с 10 элементами любых типов будет иметь начальный байт 0b100_01010 (базовый тип 4, дополнительное значение 10 указывает число элементов), за которым следует 10 элементов массива.

Базовый тип 5

Сопоставление (отображение) пар элементов данных. Отображения называют также таблицами, словарями, хэшами или объектами (в JSON). Отображение состоит из пар элементов данных, каждая из которых содержит ключ и непосредственно следующее за ним значение. Аргумент указывает число *пар* элементов в отображении. Например, отображение с 9 парами будет иметь начальный байт 0b101_01001 (базовый тип 5, дополнительное значение 9 для числа пар), за которым следует 18 элементов. Первым элементом всегда является первый ключ, вторым - первое значение, третьим - второй ключ и т. д. Поскольку элементы отображения являются парами, их число всегда чётно и отображение с нечётным числом элементов (нет значения после последнего ключа) считается некорректно сформированным. Отображение с дублированием ключей считается корректно сформированным, но непригодным и ведёт к неопределённому декодированию (см. параграф 5.6).

Базовый тип 6

Помеченный тегом элемент данных, номер тега которого указан целым числом от 0 до 2^{64}-1, включительно, являющимся аргументом, а вложенным элементом данных (*содержимое тега*) является один элемент данных, следующий за головой (см. параграф 3.4).

Базовый тип 7

Числа с плавающей точкой (запятой) и простые значения, а также код завершения break (см. параграф 3.3).

Эти 8 базовых типов задают простую таблицу, показывающую, какие из 256 возможных значений начального байта элемента данных используются (Таблица 7).

В базовых типах 6 и 7 зарезервировано множество возможных значений для будущих спецификаций (см. раздел 9).

В таблице 1 дана сводка базовых типов CBOR без учёта параграфа 3.2. Число N в таблице указывает аргумент.

Таблица 1. Сводка базовых типов CBOR с определенным размером (N - аргумент).

Базовый тип	***Значение***	***Содержимое***
0	Целое число без знака N	-
1	Отрицательное целое число -1-N	-
2	Строка байтов	N байтов
3	Строка текста	N байтов текста UTF-8
4	Массив	N элементов данных
5	Отображение	2N элементов данных (пары ключ-значение)
6	Тег с номером N	1 элемент данных
7	Простое значение, действительное число (float)	-

3.2. Неопределённый размер некоторых базовых типов

Четыре элемента CBOR (массивы, отображения, строки байтов и текста) могут кодироваться с неопределённым размером, используя дополнительное значение 31. Это полезно в случаях, когда кодирование элемента нужно начинать до того, как станет известно число элементов массива или отображения или размер строки байтов или текста (возможность начать передачу элемента данных до того, как он станет известен полностью, часто называют потоковой передачей - streaming - внутри этого элемента данных).

Массивы и отображения неопределённого размера обрабатываются иначе, чем строки неопределённого размера.

3.2.1. Код завершения break

Код завершения break представляется базовым типом 7 и дополнительным значением 31 (0b111_11111). Это не элемент данных, а просто синтаксическая функция для завершения элемента с неопределённым размером.

Если код break появляется вместо ожидаемого элемента данных, т. е. не внутри строки, массива или отображения с неопределённым размером (например, с массиве или отображении с заданным размером), вложенный элемент считается некорректно сформированным.

3.2.2. Массивы и отображения неопределённого размера

Массивы и отображения неопределённого размера представляются базовым типом с дополнительным значением 31, за которым следует (возможно пустая) последовательность элементов массива или пар ключ-значение, а затем код

завершения break (параграф 3.2.1). Иными словами, массивы и отображения неопределённого размера похожи на остальные массивы и отображения, но начинаются с дополнительного значения 31 и завершаются кодом break.

Если код break указан в отображении после ключа (вместо значения), отображение считается сформированным неверно.

Для вложенности массивов и отображений неопределённого размера не задано ограничений. Код break завершает лишь один элемент, поэтому в конструкции с вложениями число кодов break должно совпадать с числом байтов типа, начинающихся с элемента неопределённого размера (глубиной вложенности).

Предположим, например, что кодер хочет представить абстрактный массив [1, [2, 3], [4, 5]]. Кодирование с определенным размером бедет иметь вид 0x8301820203820405

```
83        -- Массив размером 3
   01     -- 1
   82     -- Массив размером 2
      02  -- 2
      03  -- 3
   82     -- Массив размером 2
      04  -- 4
      05  -- 5
```

Для каждого из 3 массивов этого элемента данных можно применить неопределённый размер, как показано ниже

```
0x9f018202039f0405ffff
9F        -- Начало массива неопределённого размера
   01     -- 1
   82     -- Массив размером 2
      02  -- 2
      03  -- 3
   9F     -- Начало массива неопределённого размера
      04  -- 4
      05  -- 5
      AH  -- break для внутреннего массива
   FF     -- break для внешнего массива

0x9f01820203820405ff
9F        -- Начало массива неопределённого размера
   01     -- 1
   82     -- Массив размером 2
      02  -- 2
      03  -- 3
   82     -- Массив размером 2
      04  -- 4
      05  -- 5
   FF     -- break

0x83018202039f0405ff
83        -- Массив размером 3
   01     -- 1
   82     -- Массив размером 2
      02  -- 2
      03  -- 3
   9F     -- Начало массива неопределённого размера
      04  -- 4
      05  -- 5
      FF  -- break

0x83019f0203ff820405
83        -- Массив размером 3
   01     -- 1
   9F     -- Начало массива неопределённого размера
      02  -- 2
      03  -- 3
      FF  -- "break"
   82     -- Массив размером 2
      04  -- 4
      05  -- 5
```

Примером отображения неопределённого размера (с двумя парами ключ-значение) может быть

```
0xbf6346756ef563416d7421ff
BF           -- Начало отображения неопределённого размера
   63        -- Первый ключ - строка UTF-8 размером 3
      46756e --   Fun
   F5        -- Первое значение - true
   63        -- Второй ключ - строка UTF-8 размером 3
      416d74 --   Amt
   21        -- Второе значение - -2
   FF        -- break
```

3.2.3. Байтовые и текстовые строки неопределённого размера

Строки неопределённого размера представляются байтом базового типа для строки байтов или текста с дополнительным значением 31, за которым могут следовать строки указанного типа (chunk — блок) с определенным размером и код завершения break (параграф 3.2.1). Элемент данных, представленный строкой неопределённого размера, является конкатенацией блоков (chunk). Если блоков нет, элемент будет пустой строкой заданного типа. Блоки нулевого размера (пустые) разрешены, но не имеют практического смысла.

Если какой-либо элемент между индикатором строки неопределённого размера (0b010_11111 или 0b011_11111) и кодом break не является строкой того же базового типа с определенным размером, строка считается неверно сформированной.

Не разрешается встраивать строки неопределённого размера в качестве блоков строки неопределённого размера. Для поддержки таких строк декодеру пришлось бы сохранять стек или хотя бы счётчик уровней вложенности. От кодера этого не требуется, поскольку внутренняя строка неопределённого размера будет состоять из блоков, которые можно просто поместить во внешнюю строку неопределённого размера.

Если какая-либо из строк текста определённого размера внутри строки с неопределённым размером недействительна, строка текста неопределённого размера будет недействительной. Это означает, что байты UTF-8 одного кода Unicode (скаляр) не могут находиться в разных блоках, т. е. блоки текстовых строк должны начинаться на границах кода.

Предположим, например, что кодированная строка имеет вид

```
0b010_11111 0b010_00100 0xaabbccdd 0b010_00011 0xeeff99 0b111_11111
5F              -- Начало строки неопределённого размера
   44           -- Строка из 4 байтов
      aabbccdd  -- Байты содержимого
   43           -- Строка из 3 байтов
      eeff99    -- Байты содержимого
   FF           -- break
```

После декодирования получается байтовая строка 0xaabbccddeeff99.

3.2.4. Сводка применения неопределённого размера в базовых типах

В таблице 2 приведена сводка базовых типов CBOR с неопределённым размером и дополнительным значением 31.

Таблица 2. Сводка базовых типов CBOR с неопределённым размером (дополнительное значение 31).

Базовый тип	***Значение***	***Вложение до кода break***
0	(неверный формат)	-
1	(неверный формат)	-
2	Строка байтов	Строки байтов определённого размера
3	Строка текста	Строки текста определённого размера
4	array	Элементы данных (массива)
5	Массив	Элементы данных (пары ключ-значение)
6	(неверный формат)	-
7	Код завершения break	-

3.3. Числа с плавающей точкой и значения без содержимого

Базовый тип 7 применяется для двух типов данных - чисел с плавающей точной и простых значений, которым не нужно содержимое. Каждое 5-битовое дополнительное значение в начальном байте имеет свой смысл, указанный в таблице 3. Подобно базовым типам для целых чисел элементы этого базового типа не включают данных содержимого и вся информация заключена в начальных байтах (голова).

Таблица 3. Дополнительные значения для базового типа 7.

5-битовое значение	***семантика***
0..23	Простое значение от 0 до 23.
24	Простое значение от 32 до 255 в следующем байте
25	IEEE 754 Half-Precision Float (следующие 16 битов)
26	IEEE 754 Single-Precision Float (следующие 32 бита)
27	IEEE 754 Double-Precision Float (следующие 64 бита)
28..30	Резерв, в настоящем документе - некорректная форма
31	Код завершение break для элементов неопределённого размера (параграф 3.2.1)

Как и для других базовых типов, 5-битовое значение 24 указывает 1-байтовое расширение - байт простого значения (для снижения путаницы применяются лишь значения 32 - 255). Это сохраняет структуру начальных байтов - как для других базовых типов размер всегда определяет дополнительное значение в первом байте. В таблице 4 указаны численные значения выделенные и доступные для простых значений.

Таблица 4. Простые значения.

Значение	***Семантика***
0..19	(не выделено)
20	false - ложь
21	true - истина
22	null - пусто
23	undefined - не определено
24..31	(резерв)
32..255	(не выделено)

Кодеру **недопустимо** вводит 2-байтовые последовательности, начинающиеся с 0xf8 (базовый тип 7, дополнительное значение 24), продолжая их байтом со значением меньше 0x20 (32). Такие последовательности некорректны. Это предполагает, что кодер не может указывать false, true, null или undefined в 2-байтовых последовательностях и корректны для них будут лишь 1-байтовые представления. В общем случае каждое простое значение имеет лишь один вариант представления.

5-битовые значение 25, 26, 27 служат для указания 16-, 32-, и 64-битовых чисел IEEE 754 [IEEE754]. Эти значения с плавающей точкой кодируются в дополнительных байтах соответвующего числа (16-битовые числа floating-point рассмотрены в Приложении D).

3.4. Теги элементов

В CBOR элемент данных может быть сопровождаться тегом для придания дополнительной семантики, однозначно определяемой номером тега. Тег имеет базовый тип 6, его аргумент (раздел 3) указывает номер, а содержимым

является один вложенный элемент данных (*содержимое тега*). Если для содержимого тега нужна дополнительная структура, она указывается вложенным элементом данных. Термин «тег» в этом документе означает весь элемент данных, включая номер и содержимое тега (данные, помеченные тегом).

Предположим, например, что строка из 12 байтов помечена тегом номер 2 для указания того, что это *bignum* без знака (параграф 3.4.3). Вложенный элемент данных будет начинаться с байта 0b110_00010 (базовый тип 6, дополнительное значение 2 - номер тега), за которым следует содержимое закодированного тега 0b010_01100 (базовый тип 2, дополнительное значение 12 для размера) и 12 байтов bignum.

В расширенной базовой модели данных определение номера тега описывает дополнительную семантику, обеспечиваемую этим номером. Семантика может включать эквивалентность некоторых помеченных элементов данных другим элементам, включая те, которые могут быть представлены в простой базовой модели данных. Например, 0xc24101 - bignum, содержимое тега которого является строкой из одного байта 0x01, эквивалентно целому числу 1, которое можно представить в виде 0x01, 0x1801 или 0x190001. Определение тега может указывать предпочтительную сериализацию (параграф 4.1), рекомендуемую для базовых кодеров, это может обеспечить предпочтение для представлений базовой модели данных по отношению к использующим тег.

Определение тега обычно указывает вложенные данные, которые подходят для тега. Определение может ограничивать содержимое тега конкретной синтаксической структурой, как это сделано для тегов, определённых в этом документе, или может задать содержимое семантически. Примером последнего является восприятие тегами 40 и 1040 разных способов представления массивов [RFC8746].

По соглашению многие теги не принимают значений null или undefined в качестве содержимого, предполагая, что вместо такого тега можно просто использовать null или undefined. В параграфе 3.4.2 дополнительно приведены некоторые соображения для одного конкретного тега по части обработки этого соглашения в прикладных протоколах и отображения на конкретные типы платформ.

От декодера не требуется понимать все номера тегов и теги могут иметь небольшое значение в приложениях, где реализация, создающая конкретный элемент данных CBOR, и реализация, декодирующая этот поток знают семантику каждого элемента в потоке данных. Основной целью тегов в этой спецификации является определение типов данных общего назначения, таких как даты. Другая цель состоит в предоставлении рекомендаций по преобразованию, когда предполагается, что элемент данных CBOR должен транслироваться в другой формат, требующий подсказки о содержимом элементов. Понимание семантики тегов не обязательно для декодера, он может просто представить номер и содержимое тега приложению без интерпретации дополнительной семантики тега.

Тег применяет семантику к вложенному в него элементу данных. Теги могут быть вложенными - если в тег A встроен тег B, который включает элемент данных C, то A применяется к резудьтату применения тега B к элементу данных C.

IANA поддерживает реестр номеров тегов, как указано в параграфе 9.2. В таблице 5 указан список номеров тегов, заданных в [RFC7049], с определениями в оставшейся части этого параграфа. Тег 35 также определён в [RFC7049] и рассмотрен в параграфе 3.4.5.3. Отметим, что после публикации [RFC7049] определено много других тегов и полный список представлен в реестре, описанном в параграфе 9.2.

Таблица 5. Номера тегов, определённые в RFC 7049.

Тег	***Элемент данных***	***Семантика***
0	Текстовая строка	Стандартная строка даты и времени, см. параграф 3.4.1
1	integer или float	Дата и время на основе эпохи, см. параграф 3.4.2
2	Строка байтов	Bignum без знака, см. параграф 3.4.3
3	Строка байтов	Отрицательное значение bignum, см. параграф 3.4.3
4	Массив	Десятичная дробь, см. параграф 3.4.4
5	Массив	Bigfloat, см. параграф 3.4.4
21	(любой)	Ожидаемое преобразование в base64url, см. параграф 3.4.5.2
22	(любой)	Ожидаемое преобразование в base64, см. параграф 3.4.5.2
23	(любой)	Ожидаемое преобразование в base16, см. параграф 3.4.5.2
24	Строка байтов	Кодированный элемент данных CBOR, см. параграф 3.4.5.1
32	Текстовая строка	URI, см. параграф 3.4.5.3
33	Текстовая строка	base64url, см. параграф 3.4.5.3
34	Текстовая строка	base64, см. параграф 3.4.5.3
36	Текстовая строка	Сообщение MIME, см. параграф 3.4.5.3
55799	(любой)	CBOR с самоописанием, см. параграф 3.4.6

Концептуально теги интерпретируются в базовой модели данных, не во время (де)сериализации. Небольшое число тегов (в данный момент 25 и 29 [IANA.cbor-tags]) зарегистрировано с семантикой, которая может требовать обработки при (де)сериализации, декодер должен знать, а кодер - контролировать точную последовательность в которой элементы данных представляются в элементе данных CBOR. Это означает, что такие теги не могут быть реализованы «поверх» базового кодера/декодера CBOR (который может не отражать порядок сериализации для элементов в отображении на уровне модели данных и обратно), поэтому реализацию этих тегов требуется встраивать в базовых кодер/декодер. Определение новых тегов с такими свойствами **не рекомендуется**.

Агентство IANA выделило номера 65535, 4294967295 и 18446744073709551615 (все 1 в формате с 16, 32 и 64 битами). Их можно применять для удобства разработчиков, которые хотят, чтобы структура данных с одним целым числом указывала наличие или отсутствие конкретного тега. Это назначение описано в разделе 10 [CBOR-TAGS]. Теги не рассчитаны на применение в реальных элементах данных CBOR и реализация **может** считать их наличие ошибкой.

Протоколы могут расширять базовую модель данных (раздел 2) элементами, представляющими момент времени, используя теги 0 и 1, целыми числами произвольного размера, используя теги 2 и 3, и значениями floating-point произвольного размера, используя теги 4 и 5.

3.4.1. Стандартная строка даты и времени

Тег 0 содержит строку текста в стандартном формате даты и времени, описанном в [RFC3339] и переопределенном в параграфе 3.3 [RFC4287], который представляет момент времени, описываемый здесь. Вложенный элемент иного типа или строка текста, не соответствующая формату [RFC4287], недействительны.

3.4.2. Дата и время на основе эпохи

Тег 1 содержит число секунд с начала эпохи (1970-01-01T00:00Z UTC) для представления гражданского времени.

Содержимое тега **должно** быть целым числом без знака (базовые типы 0 и 1) или действительным числом (floating-point, базовый тип 7 с дополнительным значением 25, 26 или 27). Иные типы недействительны.

Неотрицательные значения (базовый тип 0 и неотрицательные числа floating-point) указывают время не раньше 1970-01-01T00:00Z UTC и интерпретируются в соответствии с POSIX [TIME_T] (время POSIX называют также временем эпохи UNIX). Високосные секунды обрабатываются в POSIX особо и это приводит к 1-секундным разрывам несколько раз в десятилетие. Отметим, что приложения, которым требуется выражать время после начала 2106 г., не могут исключать поддержку 64-битовых целых чисел для содержимого тега.

Отрицательные значения (базовый тип 1 и отрицательные числа floating-point) интерпретируются в соответствии с требованиями приложения, поскольку нет универсального стандарта отсчёта секунд для UTC до 1970-01-01T00:00Z (это особенно важно для моментов времени, предшествующих разрывам в национальных календарях). Это относится и к неконечным значениям (non-finite).

Для указания долей секунд можно применять вместо целых чисел значения floating-point с тегом 1. Отметим, что это обычно требует поддержки binary64, поскольку binary16 и binary32 обеспечивают отличную от нуля дробную часть секунд лишь для короткого интервала в начале 1970-х. Приложение, которому нужна поддержка тега 1, может ограничивать содержимое тега только целыми значениями (или только floating-point).

Отметим, что типы платформ для даты и времени могут включать значение null или undefined, которые могут также быть желательны на уровне прикладного протокола. Хотя выдача тегов 1 с неконечными значениями содержимого (например, NaN для неопределённых даты и времени или Infinity для незаданной даты завершения срока) может показаться обычным способом решения задачи, использование null или undefined без тега позволяет избежать неконечных значений и сократить кодирование. Разработчикам прикладных протоколов рекомендуется рассмотреть эти ситуации и включить чёткие рекомендации по их обработке.

3.4.3. Большие числа

Протокол использует теги 2 и 3 для расширения базовой модели данных (параграф 2) типом bignum, представляющим целые числа произвольного размера. В базовой модели данных значения bignum не равны целым числам той же модели, а расширенная базовая модель, заданная этим определением тегов, определяет эквивалентность на основе численных значений и предпочтительная сериализация (параграф 4.1) никогда не использует bignum, которые можно выразить базовыми целыми числами (см. ниже).

Bignum кодируются как строка байтов, интерпретируемая как целое число n с сетевым порядком байтов. Включённые элементы иных типов недействительны. Для тега 2 значением bignum будет n, для тега 3 - -1-n. Предпочтительная сериализация строки байтов состоит в исключении начальных нулей (это означает, что для n = 0 предпочтительной сериализацией будет пустая строка байтов, см. ниже). Понимающие эти теги декодеры **должны** быть способны декодировать bignum с нулями в начале. Предпочтительная сериализация целого числа, которое можно представить базовым типом 0 или 1 состоит в таком кодировании вместо bignum (это означает, что путая строка не будет появляться в bignum при использовании предпочтительной сериализации). Отметим, что это означает, что выбор непредпочтительного представления bignum вместо базового integer при кодировании не предназначен для семантики приложения (как и выбор при более длинного представления базового integer, такого как 0x1800 для 0x00).

Например, число 18446744073709551616 (2^{64}) представляется как 0b110_00010 (базовый тип 6, тег номер 2), затем 0b010_01001 (базовый тип 2, размер 9) и 0x010000000000000000 (один байт 0x01 и 8 байтов 0x00). В шестнадцатеричной форме

```
C2                      -- тег 2
   49                   -- Строка из 9 байтов
      010000000000000000  -- Содержимое байтов
```

3.4.4. Десятичные дроби и большие действительные числа

Протоколы, применяющие тег 4, расширяют базовую модель данных элементами, представляющими десятичные дроби произвольного размера в форме $m*10^e$. Протоколы, применяющие тег 5, расширяют базовую модель данных элементами, представляющими двоичные дроби произвольного размера в форме $m*2^e$. Как и для bignum, значения разных типов не эквивалентны в базовой модели без расшинения.

Десятичная дробь - это целочисленная мантисса (m) и «масштабный коэффициент» в форме степени числа 10. такие дроби наиболее полезны в приложениях, требующих точного представления десятичной дроби, такой как 1,1, поскольку в двоичном формате с плавающей точкой нет точного представления для многих десятичных дробей.

Bigfloat - это целочисленная мантисса (m) и «масштабный коэффициент» в форме степени числа 2. Это двоичное значение с плавающей точкой, выходящее за пределы диапазона или точности трёх форматов IEEE 754, поддерживаемых CBOR (параграф 3.3). Bigfloat можно также применять в приложениях с ограничениями, где нужна некоторая поддержка действительных чисел без потребности поддерживать IEEE 754.

Десятичные и двоичные дроби представляются как массив с тегом, содержащий два целых числа. - показатель e и мартиссу m. Десятичные дроби (тег 4) используют основание 10 и значением элемента данных является $m*10^e$. Bigfloat (тег 5) использует основание 2 и имеет значение $m*2^e$. Показатель e **должен** представляться как целое число типа 0 или 1, а мантиссой может быть bignum (параграф 3.4.3). Элементы с иной структурой не действтительны.

Примером десятичной дроби может служить представление числа 273,15 как 0b110_00100 (базовый тип 6 с тегом 4), затем 0b100_00010 (базовый тип 4 для массива с дополнительным значением 2 для размера массива), 0b001_00001 (базовый типа 1 для первого целого числа, дополнительное значение 1 для -2), 0b000_11001 (базовый тип 0 для второго целого числа, дополнительное значение 25 для 2 байтов) и 0b0110101010110011 (27315 в двух байтах). В шестнадцатеричном формате это будеи

```
C4          -- Тег 4
   82       -- Массив размером 2
```

```
      21       -- -2
      19 6ab3  -- 27315
```

Примером bigfloat может служить представление числа 1,5 как 0b110_00101 (базовый тип 6 для тега и номер тега 5), хатем 0b100_00010 (базовый тип 4 для массива и дополнительное значение 2 для размера), 0b001_00000 (базовый тип 1 для первого целого числа и дополнительное значение 0 для числа -1) и 0b000_00011 (базовый тип 0 для второго целого числа и дополнительное значение 3 для числа 3). Шестнадцатеричное представление имеет вид

```
C5            -- Тег 5
   82         -- Массив размером 2
      20      -- -1
      03      -- 3
```

Десятичные дроби и bigfloat не обеспечивают представление Infinity, -Infinity, NaN. Если они нужны, можно применять взамен представление IEEE 754 с половинной точностью, как описано в параграфе 3.3.

3.4.5. Подсказки содержимого

Описываемые здесь теги предназначены для подсказки содержимого, которую могут использовать базовые процессоры CBOR. Эти подсказки не расширяют базовую модель данных.

3.4.5.1. Закодированный элемент данных CBOR

Иногда полезно передать встроенный элемент данных CBOR, не предназначенный для декодирования вместе с включающим его элементом данных. Тег 24 (элемент данных CBOR) можно применять для маркировки встроенной строки байтов как одного элемента, кодированного в формате CBOR. Не являюшиеся строками байтов элементы недействительны. Включённая строка байтов действительна, если она кодирует корректно сформированный элемент данных CBOR, проверка пригодности декодированного элемента CBOR не требуется для валидности тега (но может предлагать базовым декодером как специальная опция).

3.4.5.2. Ожидаемое последующее кодирование для преобразования CBOR в JSON

Теги 21 - 23 указывают, что строка байтов может требовать конкретного кодирования при взаимодействии с основанным на тексте представлением. Эти теги полезны, когда кодер знает, что записываемая строка байтов позднее будет преобразовываться для определённого использования на основе JSON. Такое использование указывает, что некоторые строки закодированы как base64, base64url и т. п. Кодер применяет строки байтов вместо выполнения самого кодирования с целью снизить размер кода, самого кодера или обоих. Кодер не знает, будет ли преобразователь базовым и поэтому хочет сказать, что он надеется на корректное преобразование двоичных строк в JSON.

Помеченный элемент данных может быть строкой байтов или иным элементом данных. В последнем случае тег применяется ко всем элементам данных строки байтов, содержащимся в элементе данных, за исключением тех, которые находятся во вложенном элементе, помеченном ожидаемым преобразованием.

Эти три тега предполагают преобразование в три базовых варианта кодирования, определённые с [RFC4648]. Тег 21 предполагает преобразование в base64url (раздел 5 в [RFC4648]) без заполнения (см. параграф 3.2 в [RFC4648]), т. е. все завершающие символы = (равенство) удаляются из кодированной строки. Тег 22 предполагает преобразование в классический код base64 (раздел 4 в [RFC4648]) с заполнением в соответствии с RFC 4648. Для base64url и base64 используются биты заполнения 0 (см. параграф 3.5 в [RFC4648]) и выполняется преобразование для дополнительного кодирования над содержимым строки байтов (т. е. без добавления символов завершения строки, пробелов и других дополнительных символов). Тег 23 предполагает преобразование в base16 (hex) с заглавными буквами (см. раздел 8 в [RFC4648]). Отметим, что для всех тегов кодирование пустой строки байтов даёт пустую строку текста.

3.4.5.3. Кодированный текст

Некоторые строки текста содержат данные в форматах, широко применяемых в Internet, а иногда декодер может проверить эти форматы и представить приложению в подходящей форме. Для части форматов имеются теги.

- Тег 32 применяется для URI, определённых в [RFC3986]. Если строка текста не соответствует URI-reference, она будет недействительной.
- Теги 33 и 34 служат для текста, закодированного в base64url или base64, соответственно, как задано в [RFC4648]. Строки, соответствующие любому из приведённых ниже условий, являются недействительными:
  - строка кодированного текста содержит неалфавитные символы или только 1 алфавитный символ в последнем из 4 блоков (алфавит определён в разделе 5 [RFC4648] для тега 33 и в разделе 4 [RFC4648] для тега 34);
  - биты заполнения в 2-х или 3-символьном блоке отличны от 0;
  - кодирование base64 имеет некорректное число символов заполнения;
  - кодирование base64url включает символы заполнения.
- Тег 36 служит для сообщений MIME (включая все заголовки), определённых в [RFC2045]. Строка текста, не являющаяся корректным сообщением MIME, будет недействительна. Для этого тега проверка корректности может быть особенно обременительной для базовых декодеров и по этой причине может не предоставляться. Отметим, что многие сообщения MIME представляют собой обычные двоичные данные и поэтому не могут быть представлены текстовыми строками. В [IANA.cbor-tags] представлена регистрация тега 257, похожего на тег 36, но использующего в качестве содержимого строку байтов.

Отметим, что теги 33 и 34 отличаются от тегов 21 и 22 тем, что данные передаются первыми в форме base-кода, а вторыми - в raw-строк байтов.

В [RFC7049] определён тег 35 для регулярных выражений Perl в форме PCRE/PCRE2 (Perl Compatible Regular Expression) [PCRE] или JavaScript [ECMA262]. Методы регулярных выражений с момента публикации документов были усовершенствованы и данная спецификация не пытается обновить ссылки. Тег остаётся доступным (как зарегистрировано в [RFC7049]) для приложений, которые задают конкретный вариант регулярного выражения

(возможно с ограничением определенным подмножеством PCRE и ECMA262). Поскольку эта спецификация уточняет пригодность тегов сверх [RFC7049], отмечается, что в результате открытого способа определения тега в [RFC7049], любая содержащаяся в нем строка должна быть действительной на уровне тега CBOR (но это может не совпадать с ожиданием прикладного уровня).

3.4.6. Самоописание CBOR

Во многих приложениях из контекста понятно применение CBOR для кодирования элементов данных. Например, конкретный протокол может указывать использование CBOR или для типа носителя указано такое применение. Однако возможны приложения, где такой контекстной информации нет, например, при хранении данных CBOR в файле без метаданных, устраняющих неоднозначность. Здесь может помочь наличие отличительных признаков в самих данных.

Для этого определён тег 55799, применяемый, в частности, в начале сохраняемых данных в коде CBOR, как указано приложением. Это не задаёт особой семантики элемента данных, т. е. семантика содержимого тега с номером 55799 является семантикой самого содержимого.

Сериализация головы тега будет иметь вид 0xd9d9f7, что, похоже, не служит отличительным признаком для каких-либо часто используемых файлов. В частности, 0xd9d9f7 не является дозволенным началом текста Unicode в какой-либо кодировке Unicode, если за ним следует действительный элемент данных CBOR.

Например, декодер может быть способен декодировать как CBOR, так и JSON. Такому декодеру нужно различать два этих формата. Простым способом помочь этому со стороны кодера было бы указание всего элемента CBOR с тегом 55799, сериализация которого никогда не будет присутствовать в тексте JSON.

4. Вопросы сериализации

4.1. Предпочтительная сериализация

На уровне модели для некоторых значений CBOR представляет несколько вариантов сериализации. Во многих приложениях желательно, чтобы кодер всегда выбирал предпочтительную сериализацию (кодирование), однако данная спецификация не требует этого ни от кодера, ни от декодера.

Некоторые декодеры с ограничениями могут быть ограничены в способности декодировать отличную от предпочтительной сериализацию, например, если приложение ожидает лишь целые числа, не превышающие 1 миллард (1 000 000 000), декодер может не включать кода, требуемого для 64-битовых целочисленных аргументов. Кодер, который должен всегда применять предпочтительную сериализацию (кодер с предпочтением), будет совместим с таким декодером для чисел, встречающихся в приложении. Вообще говоря, кодер с предпочтениями более универсален в части взаимодействия (и менее расточителен), нежели кодер, который всегда применяет, скажем, 64-битовые целые числа.

Точно так же кодер с ограничениями может быть ограничен в выборе поддерживаемых вариантов представления и не сможет выдавать предпочтительную сериализацию (кодер варианта). Например, кодер с ограничениями может быть спроектирован так, что всегда применяется 32-битовый вариант кодирования целых чисел даже при доступности более короткого представления (в предположении отсутствия потребности в 64-битовом представлении). Декодер, который не полагается на приём лишь предпочтительной сериализации (устойчивый к вариантам декодер) целых чисел можно считать более универсальным по части совместимости (хотя он может быть оптимизирован для получения предпочтительной сериализации). Полные реализации декодеров CBOR по определению устойчивы к вариациям и различие важно лишь в случае взаимодействия ограниченной реализации декодера CBOR с кодером варианта.

Предпочтительная сериализация всегда использует кратчайшую форму представления аргумента (раздел 3), а также кратчайшее кодирование floating-point, сохраняющее кодируемое значение.

Предпочтительной сериализацией действительного числа (floating-point) является кратчайшее кодирование floating-point, сохраняющее значение числа, например, 0xf94580 для числа 5,5 и 0xfa45ad9c00 для 5555,5. Для зачений NaN предпочтительно более короткое кодирование, если дополнение нулями со сдвигом вправо восстанавливает исходное значение NaN (для многих приложений достаточно одного кодирования NaN - 0xf97e00).

Кодирование с определенным размером предпочтительно, когда размер известен при старте сериализации элемента.

4.2. Детерминированное кодирование CBOR

Некоторым протоколам могут потребоваться кодеры, которые выдают CBOR только в детерминированном формате, эти протоколы могут также иметь декодеры, проверяющие входной формат на предмет соответствия заданному. Такие протоколы могут сами определять, что они считают «определенным форматом» и что они ждут от кодеров и декодеров В этом разделе задан набор ограничений, которые могут служить базой для определённого формата.

4.2.1. Базовые требования детерминированного кодирования

Кодирование CBOR удовлетворяет «базовым требованиям детерминированного кодирования» при выполнении перечисленных ниже условий.

- **Должна** применяться предпочтительная сериализация. В частности, это означает, что аргументы (раздел 3) для целых чисел, размеры базовых типов 2 - 5 и теги **должны** быть максимально короткими, например:
  - значения от 0 до 23 и от -1 до -24 **должны** выражаться в одном байте с базовым типом;
  - значения от 24 до 255 и от -25 до -256 **должны** выражаться только дополнительным полем uint8_t;
  - значения от 256 до 65535 и от -257 до -65536 **должны** выражаться только дополнительным полем uint16_t;
  - значения от 65536 до 4294967295 и от -65537 до -4294967296 **должны** выражаться только дополнительным полем uint32_t.

Значения с плавающей точкой тоже **должны** использовать кратчайшую форму, сохраняющую значение, например 1,5 кодируется как 0xf93e00 (binary16), а 1000000.5 - как 0xfa49742408 (binary32). Одна из реализаций этого заключается в использовании сначала 64-битового формата, а затем тестового преобразования в 32-битовый и при совпадении значений - тестового преобразования в 16-битовый формат. Это также работает для 16-битовых чисел с плавающей точкой для положительных и отрицательных значений Infinity.

- Значения неопределённого размера **недопустимо**, но их можно кодировать элементами определённого размера.
- Ключи каждого отображения **должны** сортироваться в побайтовом лексикографическом порядке их детерминированных кодировок, ниже приведён пример корректной сортировки.

  1. 10, закодированный как 0x0a;
  2. 100, закодированный как 0x1864;
  3. -1, закодированный как 0x20;
  4. "z", закодированный как 0x617a;
  5. "aa", закодированный как 0x626161;
  6. [100], закодированный как 0x811864;
  7. [-1], закодированный как 0x8120;
  8. false, закодированный как 0xf4.

Примечание для разработчиков. Саморазграничение кодировок CBOR означает, что не может двух корректно сформированных кодированный элементов данных CBOR, из которых один является префиксом другого. Таким образом, побитовое лексикографическое сравнение детерминированных кодировок разных ключей отображения всегда находит различие между ключами до того, как будет достигнут конец любого из сравниваемых ключей.

4.2.2. Дополнительные вопросы детерминированного кодирования

Теги CBOR представляют дополнительные соображения по части детерминированного кодирования. Если протокол на основе CBOR обеспечивает одинаковую семантику независимо от наличия или отсутствия конкретного тега (например, разрешая элементы данных с тегом 1 и необработанные (raw) числа для даты и времени, трактуя последние как при наличии тега), детерминированный формат не допустит наличия тега из-за принципа использования кратчайшей формы. Например, протокол может предоставлять кодеру выбор представления URL в форме строки текста или с помощью тега 32 (параграф 3.4.5.3), содержащего строку текста. Детерминированное кодирование этого протокола должно требовать наличия или отсутствия тега, не допуская выбора.

В протоколе, требующем теги в некоторых местах для обеспечения конкретной семантики, теги должны присутствовать в детерминированном формате. Требования к детерминированному кодированию относятся и к содержимому тегов.

Если протокол включает поле, которое может содержать целое число со значением 2^{64} или больше с использованием тега 2 или 3 (параграф 3.4.3), детерминированное кодирование протокола должно указывать, выражаются ли меньшие числа с использованием тех же тегов или могут использовать базовые типы 0 и 1. Предпочтительная сериализация применяет последний вариант и поэтому рекомендуется.

Протоколы, включающие действительные (floating-point) значения, независимо от применения базовых значений floating-point (параграф 3.3) или тегов, могут требовать задания дополнительных требования к детерминированному кодирования, как показано ниже.

- Хотя значения IEEE floating-point могут представлять положительные и отрицательные значения, приложение может не разделять их и может представлять значение, содержащее только 0, как положительное, запрещая «отрицательный 0». Приложение может также ограничивать точность floating-point, чтобы никогда не применялись 64-битовые или даже 32-битовые значения с плавающей точкой.
- Если протокол включает поля, которые могут выражать значения floating-point, с конкретной моделью данных, заявляющей целые и действительные числа невзаимозаменяемыми, детерминированное кодирование протокола должно указывать, например, что число 1,0 кодируется как 0x01 (целое без знака), 0xf93c00 (binary16), 0xfa3f800000 (binary32), or 0xfb3ff0000000000000 (binary64). Примерные варианты даны ниже.

  1. Кодировать целые числа, требующие 64 бита базовыми типами 0 или 1, а иные значения в соответствии с предпочтительным форматом floating-point, сохраняющим точность (меньшее из 16, 32 или 64 битов).

  1. Кодировать все значения с предпочтительным представлением floating-point, сохраняющим точность, даже для целых чисел.
  2. Кодировать все значения в 64-битовом представлении с плавающей точкой.

  Правило 1 ставит границу между целочисленными и действительными значениями, в правило 3 не применяет предпочтительную сериализацию, поэтому во многих случаях лучше применять правило 2.

- Если разрешены значения NaN и нет намерений поддерживать NaN в данных (payload) или сигнализации, протокол должен указать обно представление (обычно 0xf97e00). Если этот простой вариант не приемлем, нужно обратить особое внимание на обработку NaN.
- Субнормальные числа (ненулевые значения с минимальным показателем для данного формата IEEE 754) могут отсекаться на выходе до 0 или трактоваться как 0 в некоторых реализациях floating-point. Детерминированное кодирование протокола может специально приспособить такие реализации, перенося нагрузку на другие реализации, исключая субнормальные числа из обмена и используя вместо них 0.

- Одно и то же число можно представить разными десятичными дробями, двоичными дробями (bigfloat) и иными формами с тегами, которые могут быть определены для числовых значений. В зависимости от реализации не всегда практично задавать, какие из этих форм (или форм базовой модели данных) эквивалентны. Прикладной протокол, предоставляющий выбор варианта представления чисел, должен явно указывать, как это делается для детерминированного формата.

4.2.3. Упорядочение ключей сначала по размеру

Базовые требования детерминированного кодирования (параграф 4.2.1) задают сортировку ключей в порядке, отличающемся от заданного в параграфе 3.9 [RFC7049] (здесь он называется каноническим - Canonical CBOR). Протоколы, которым нужна совместимость с [RFC7049], могут задать в терминах этой спецификации сортировку сначала по размеру ключа (length-first core deterministic encoding requirements). В этом случае **должны** выполняться приведённые ниже правила.

1. Из двух ключей разного размера первым считается более короткий.
2. Из двух ключей одного размера первым считается тот, у которого меньше побайтовое лексикографическое значение.

Например, в соответствии с этими требованиями ключи будут сортироваться в приведённом ниже порядке.

1. 10, закодированный как 0x0a;
2. -1, закодированный как 0x20;
3. false, закодированный как 0xf4;
4. 100, закодированный как 0x1864;
5. "z", закодированный как 0x617a;
6. [-1], закодированный как 0x8120;
7. "aa", закодированный как 0x626161;
8. [100], закодированный как 0x811864.

Хотя в [RFC7049] применяется термин Canonical CBOR для указания требований к детерминированному кодированию, данный документ избегает этого, поскольку «канонизация» часто ассоциируется только с конкретным применением детерминированного кодирования. Однако термины по существу взаимозаменяемы и набор базовых требований этого документа также можно назвать «каноническим CBOR», а вариант с сортировкой сначала по размеру - Old Canonical CBOR.

5. Создание протоколов на основе CBOR

Такие форматы данных как CBOR часто применяются в средах без согласования формата. Одной из целей CBOR является исключение потребности в какой-либо включённой или предполагаемой схеме - декодер может взять элемент CBOR и декодировать его без дополнительных сведений.

В практических реализациях конечно требуется, чтобы кодер и декодер имели общее представление об элементах данных CBOR. Например, согласованным форматом может быть «элемент - это массив, где первым значением является строка UTF-8, вторым - целое число, а последующие элементы могут содержать значения floating-point» или «элемент - это отображение с байтовыми строками для ключей и парой с ключом 0xab01».

Протоколы на основе CBOR **должны** задавать, как их декодеры обрабатывают недействительные или неожиданные данные. Протокол на основе CBOR **может** задавать трактовку произвольных действительных данных как неожиданных. Кодеры протоколов на основе CBOR **должны** создавать только действительные элементы, т. е. протокол не может использовать недействительные элементы. Кодер может быть способен кодировать столько типов значений, сколько требуется протоколу, где кодер применяется. Декодер может быть способен понимать столько типов значений, сколько требуется применяющему его протоколу. Такое отсутствие ограничений позволяет применять CBOR в разных средах.

В остальной части этого раздела рассматриваются некоторые вопросы создания протоколов на основе CBOR. За некоторыми исключениями, это будут лишь рекомендации и явно исключаются уровни требований BCP 14 [RFC2119] [RFC8174], кроме уровня **можно**, интерпретируемого в соответствии с BCP 14. Исключения нацелены на облегчение совместимости протоколов на основе CBOR с использованием широкого спектра базовых и зависимых от приложений кодеров и декодеров.

5.1. CBOR в потоковых приложениях

В потоковом приложении данные могут быть последовательностью элементов CBOR, соединённых один с другим (конкатенация). В таких случаях декодер начинает декодирование элемента данных сразу после обнаружения конца предыдущего элемента. Не все байты элемента данных могут быть сразу доступны декодеру, некоторые декодеры буферизуют данные до представления элемента приложению целиком. Другие декодеры могут представлять приложению часть информации об элементе данных верхнего уровня, такую как сведения о вложенных элементах, которые уже можно декодировать, или даже часть строки байтов, которая ещё не получена целиком. Такие приложения **должны** иметь для потока соответствующий механизм защиты данных, предоставляемых приложению постепенно.

Отметим, что некоторые приложения и протоколы могут отказаться от применения элементов неопределённого размера. Кодирование таких элементов позволяет кодеру не выстраивать все данные для подсчёта размера, но требует от декодера выделять больше памяти для буферизации элемента. Это удобно не во всех приложениях.

5.2. Базовые кодеры и декодеры

Базовый декодер CBOR может декодировать и представлять приложению все корректно сформированные элементы данных CBOR (см. Приложение C). для представления корректно сформированных значений CBOR человеку можно применять диагностическую нотацию (раздел 8).

Базовый кодер CBOR предоставляет приложению интерфейс, позволяющий указать для кодирования в элемент данных CBOR любое корректно сформированное значение, включая простые значения и теги, неизвестные кодеру.

Хотя CBOR пытается минимизировать случаи недействительности корректно сформированных данных CBOR, он возможны. Например, кодированная строка текста "0x62c0ae" не содержит пригодного UTF-8 (поскольку [RFC3629] требует использовать кратчайшую форму) и поэтому не является действительным элементом CBOR. Кроме того, конкретные теги могут вносить семантические ограничения, которые могут нарушаться, например, тег bignum, включающий другой тег, или экземпляр тега 0 со строкой байтов или текста, не соответствующей дате и времени [RFC3339]. От базовых кодеров и декодеров не требуется делать неестественный выбор для своих приложений с целью обработки недействительных данных. Предполагается, что базовые кодеры и декодеры пересылают простые значения и теги, даже если их конкретные коды не были зарегистрированы на момент создания кодера или декодера (параграф 5.4).

5.3. Пригодность элементов

Корректно сформированный, но недействительный элемент данных CBOR (параграф 1.2) создаёт проблему при интерпретации данных, закодированных в модели данных CBOR. Протокол на основе CBOR может включать несколько уровней и нижние уровни не смогут обработать семантику некоторых пересылаемых данных CBOR. Эти уровни не могут проверить пригодность данных, которые они обрабатывают, и **должны** пересылать их как есть. Первый уровень, обрабатывающий семантику непригодного элемента CBOR должен выбрать один из двух вариантов:

1. заменить проблемный элемент маркером ошибки и продолжить обработку следующего элемента;
2. выдать ошибку и прекратить обработку.

Протокол на основе CBOR **должен** указать какой из этих вариантов его декодеры выбирают для каждого встречающегося непригодного элемента. Отмеченная проблемы могут возникать на базовом уровне проверки пригодности CBOR или в контексте тегов (пригодность тега).

5.3.1. Базовая пригодность

На уровне базовой модели данных возможны две ошибки пригодности.

Дубликаты ключей в отображении
: Базовые декодеры (параграф 5.2) делают данные доступными для приложений на основе естественной модели данных CBOR. Эта модель включает отображения (пары ключ-значение с уникальными ключами), но не множественные отображения (multimap, гда одному ключу может соответствовать несколько записей). Таким образом, базовый декодер, получающий элемент отображения CBOR с дубликатом ключа будет декодировать лишь один экземпляр и прекращать дальнейшую обработку. Кроме того, «потоковый декодер» может не заметить дубликата. Более подробное рассмотрение ключей отображений приведено в параграфе 5.6.

Неприголные строки UTF-8
: Декодер может проверять или не проверять фактическую допустимость байтов в строке UTF-8 (базовый тип 3) с точки зрения корректности UTF-8 и в любом случае соответствующим образом реагировать.

5.3.2. Пригодность тегов

Два дополнительных типа ошибок пригодности связаны с добавлением тегов в базовую модель данных.

Недопустимое для типа тега содержимое
: Номера тегов (параграф 3.4) указывают тип элемента данных, служащего содержимым тега, например, теги для беззнаковых или отрицательных bignum предполагают битовые строки. Декодер преобразует помеченный элемент данных в естественное представление (в данном случае естественное большое число) и от него ожидается проверка типа данных в теге. Даже декодеры, не имеющие в своей среде доступного представления, могут проверять известные им теги и реагировать должным образом.

Недопустимое для типа тега значение
: Для элемента данных с тегом пригодный тип может содержать недействительное значение, например, "yesterday" не подходит в качестве содержимого тега 0, хотя и является корректной строкой текста. Декодер, который обычно преобразует такие значения в подходящий для платформы эквивалент, может представить тег приложению так же, как он представил бы тег с неизвестным номером (параграф 5.4).

5.4. Пригодность и развитие

Декодер с проверкой пригодности затрачивает усилия для надёжного обнаружения элементов данных с ошибками пригодности. Например, такому декодеру нужно иметь API для уведомления об ошибках (без возврата данных) для элементов CBOR, содержащих любую из указанных выше ошибок пригодности.

Набор тегов из реестра Concise Binary Object Representation (CBOR) Tags (параграф 9.2), а также простых значений из реестра Concise Binary Object Representation (CBOR) Simple Values (параграф 9.1) может быть расширен в любой момент так, что базовый декодер не будет знать новых значений. Декодер с проверкой пригодности в таких случаях может выбрать один из указанных ниже вариантов.

- Сообщить об ошибке (не возвращая данных). Отметим, что это может приводить к «окостенению» и по этой причине не рекомендуется. Такая ошибка сама по себе не является ошибкой пригодности и скорей всего выдаётся декодером, выполняющим проверку пригодности из-за неизвестного тега или простого значения.
- Передать неизвестный элемент (тип, значение, а для тегов и декодированный элемент) вызвавшему декодер приложению, а затем указать ему, что тег или простое значение не распознаны.

Во втором варианте, подходящем и для декодеров без проверки пригодности, обеспечивается совместимость с новыми тегами и простыми значениями без необходимости обновлять кодер вместе с вызывающим приложением. Для этого API декодера должен быть способен помечать неизвестные элементы, чтобы вызывающее приложение могло обработать их в соответствии с потребностями программы.

Поскольку обработка при проверке пригодности может быть связана со значительными издержками (в частности для обнаружения дубликатов в отображениях), поддержка проверки пригодности не требуется во всех декодерах CBOR.

Некоторые кодеры могут полагаться на свои приложения для предоставления входных данных так, чтобы кодер выдавал действительные результаты CBOR. Базовый кодер может также обеспечивать режим проверки пригодности, в котором он надёжно ограничивает свой вывод лишь действительными элементами CBOR, независимо от представления приложением совместимых с API данных.

5.5. Числа

Протоколам на основе CBOR следует учитывать, что разные языковые среды вносят разные ограничения на диапазоны и точность представляемых чисел. Например, базовая система счисления JavaScript считает все числа действительными (floating-point), что может вести к скрытой потере точности для целых чисел с числом значащих битов более 53. Поскольку CBOR сохраняет бит знака для своего представления целых чисел в базовом типе, имеется ещё один бит для чисел со знаком определённого размера (например, $-2^{64}..2^{64}-1$ для 1+8-байтовых целых чисел) по сравнению с обычным для платформы представлением целых чисел со знаком и таким же размером ($-2^{63}..2^{63}-1$ для 8-битового int64_t). Протоколу, использующему числа, следует указывать свои ожидания в части обработки нетривиальных чисел в декодерах и принимающих приложениях.

Протоколы на основе CBOR, использующие действительные числа могут ограничивать набор поддерживаемых форматов (половинная, одинарная и двойная точность). Для приложений, применяющих лишь целые числа, протокол может полностью исключить значения floating-point.

Протоколы CBOR, предназначенные для обеспечения компактности, могут исключать определённые варианты кодирования целых чисел, которые не нужны для приложений, например, требующие 64-битовых форматов. Предполагается, что кодеры будут использовать наиболее компактное представление целых чисел, подходящее для данного значения. Однако компактным приложениям, не требующим детерминированного кодирования, следует воспринимать значения, использующие представление с размером больше необходимого (например, представление 0 в виде 0b000_11001 с двухбайтовым значением 0x00), если приложение может декодировать целое число такого размера. Подобные соображения применимы и для действительных чисел - рекомендуется декодировать как предпочтительную сериализацию, так и более длинные форматы.

Протоколы на основе CBOR для приложений с ограничениями, которые предоставляют выбор между представлением данного значения целым числом, десятичной или двоичной (bigfloat) дробью (например, когда показатель мал и неотрицателен), могут выражать ожидание реализации, где целые числа представляются напрямую.

5.6. Задание ключей для отображений

Приложения кодирования и декодирования должны согласовывать типы ключей, используемых в отображениях. В приложениях, которым нужно взаимодействовать с приложениями на основе JSON преобразование упрощается применением лишь ключей в форме строк текста, в противном случае потребуется конкретное отображение других типов CBOR в строки текста, а это часто ведёт к ошибкам реализации. В приложениях, где ключи числовые по своей природе, полезно напрямую применять численные ключи.

Если требуется несколько типов ключей, следует рассмотреть применение этих типов в конкретных программных средах, которые будут использоваться. Например, в JavaScript Maps [ECMA262], целочисленный ключ 1 не отличить от ключа 1,0 (floating-point). Это значит, что при использовании целочисленных ключей протоколу потребуется избегать ключей floating-point со значениями, совпадающими с имеющимися в отображении целочисленными ключами.

Декодеры, доставляющие элементы данных, вложенные в элемент данных CBOR, сразу после их декодирования (потоковые декодеры). зачастую не сохраняют состояние, требуемое для проверки уникальности ключа в отображении. Точно так же кодер, который может начинать кодирование элемента данных до полной доступности вложенного элемента (потоковый кодер), может снижать свои издержки, полагаясь в плане уникальности ключей на источник.

Протоколы на основе CBOR **должны** указывать, что делать, когда принимающее приложение видит в отображении совпадающие ключи. Результирующее правило в протоколе **должно** соответствовать модели данных CBOR, оно не может предписывать конкретную обработку записей с одинаковыми ключами, за исключением возможности иметь правило, по которому наличие идентичных ключей указывает некорректную форму отображения и позволяет декодеру остановить обработку с возвратом ошибки. При обработке отображений, содержащих записи с одинаковыми ключами, базовый декодер может выбрать один из указанных ниже вариантов.

- Не отвергать отображения с дубликатами ключей (считать их пригодными, см. параграф 5.4). Такие базовые декодеры полезны. Приложение может быть заинтересовано в выполнении своей проверки правил применимости (например, если оно считает эквивалентными целые и действительные значения ключей для конкретных отображений).
- Передавать приложению все записи отображения, включая дубликаты ключей. Это требует от приложения обработки (проверки) дубликатов, даже если правила приложения идентичны правилам базовой модели.
- Исключать записи с дубликатами ключей, доставляя, например, только последнюю (или первую) из них. С такими базовыми декодерами приложение может получать разные результаты для конкретного ключа при разных запусках или при использовании разных декодеров, которые возвращают значение с учётом реализации и фактического порядка ключей в отображении. В частности, приложения не могут сами проверить уникальность ключей, поскольку они не обязательно получают все записи, они могут быть неспособны применять такой базовый декодер, если требуется проверка уникальности ключей. Такие базовые декодеры можно применять лишь в ситуациях, когда источник и отправитель данных всегда представляют пригодные отображения, что невозможно, если источник или отправитель могут быть атакованы.

Базовые декодеры должны указывать в документации используемый подход из числа перечисленных выше.

Модель данных CBOR для отображений не позволяет приписать семантику порядку пар ключ-значение в представлении отображения. Поэтому основанному на CBOR протоколу **недопустимо** задавать изменение семантики при смене порядка пар ключ-значение за исключением запрета некоторых вариантов порядка, например, при нарушении правил детерминированного кодирования (параграф 4.2). Вторичные эффекты упорядочения отображений, такие как синхронизация, использование кэша и другие возможные побочные влияния не считаются частью семантики, но могут служить достаточным основанием для того, чтобы протокол требовал детерминированного кодирования.

Приложениям для устройств с ограничениями следует рассмотреть применение небольших целых чисел в качестве часто используемых ключей, например, набор из 24 (или меньше) ключей можно закодировать одним байтов как целое число без знака, а использование отрицательных значений позволяет расширить набор до 48 ключей. Для более редко применяемых ключей можно использовать целые числа большего размера.

5.6.1. Эквивалентность ключей

Для определения дубликатов ключей отображения служит конкретная модель данных, применяемая к элементу CBOR.

В базовой модели данных целые и действительные числа с одним значением считаются разными, поскольку они происходят от разных больших чисел (теги 2 - 5). Точно так же строки текста отличаются от строк байтов, даже если они содержат одинаковые байты. Значение с тегом отличается от значения без тега или значения с иным тегом.

В каждой из этих групп численные значения различаются, если они не совпадают по величине (в частности, -0,0 = 0,0) при определении дубликатов ключей в отображении. Значения NaN эквивалентны, если они имеют одинаковые значения обеих частей после их дополнения справа нулями до 64 битов.

Строки байтов и текста сравниваются побайтно, а массивы - поэлементно и считаются равными, если имеют одинаковое число байтов или элементов и одинаковые значения в тех же позициях. Отображения являются равными, если они имеют одинаковые наборы пар независимо от их упорядочения, пары считаются равными при совпадении ключа и значения в них.

Значения с тегами равны, если совпадают номера и содержимое тегов. Отметим, что базовый декодер, обрабатывающий конкретный тег, может не различать некотрые семантически эквивалентные значения, например, нули слева в содержимом с тегом 2 или 3 (параграф 3.4.3). Простые значения равны при их совпадении. В базовой модели данных нет ничего равного, простое значение 2 не эквивалентно целочисленному значению 2, а массив никогда не эквивалентен отображению.

Как обсуждалось в параграфе 2.2, конкретная модель данных может считать при сравнении ключей эквивалентными значения, которые различаются в базовой модели данных. Отметим, что это предполагает, что базовый декодер может достаить приложению декодированное отображение, которое приложение должно проверить на предмет дубликатов ключей (декодер может также предоставить приложению программный интерфейс для выполнения этой задачи). Конкретные модели данных не будут различать в отображении ключи, одинаковые на уровне базовой модели данных.

5.7. Неопределённые значения

В некоторых протоколах на основе CBOR простое значение undefined (параграф 3.3) может применяться кодером для подстановки вместо вызывающих проблему элементов данных, чтобы закодировать остальные элементы без вреда.

6. Преобразование данных между CBOR и JSON

В этом разделе приведены ненормативные советы по преобразованию между CBOR JSON. Реализации конвертеров могут применять эти рекомендации по своему усмотрению. Следует отметить, что текст JSON является последовательностью символов, а не байтов, как в CBOR.

6.1. Преобразование CBOR в JSON

Большинство типов CBOR имеет прямые аналоги в JSON. Однако аналоги есть не всегда и при внедрении конвертеров CBOR в JSON следует учитывать это. Ниже приведены ненормативные рекомендации по предобразованию в одно подстановочное значение, такое как JSON null.

- Целые числа (базовые типы 0 и 1) становятся числами JSON.
- Строка байтов (базовый тип 2), не встроенная в тег, задающий предлагаемое кодирование, преобразуется в base64url без дополнение и становится строкой JSON.
- Строка UTF-8 (базовый тип 3) становится строкой JSON. Отметим, что JSON требует экранирования некоторых символов (раздел 7 в [RFC8259]): кавычки (U+0022), обратная дробная черта (U+005C) и символы управления C0 (U+0000 - U+001F). Остальные символы копируются в строку JSON UTF-8 без изменений.
- Массив (базовый тип 4) становится массивом JSON.
- Отображение (базовый тип 5) становится объектом JSON. Это можно сделать напрямую лишь при использовании в качестве ключей строк UTF-8. Конвертер может преобразовать другие ключи в строки UTF-8 (например, целые числа в десятично представление), однако это создаёт риск дублирования ключей. Отметим, что при игнорировании тегов в строках UTF-8 (см. ниже) могут возникать конфликты ключей.
- False (базовый тип 7, дополнительное значение 20) становится JSON false.
- True (базовый тип 7, дополнительное значение 21) становится JSON true.
- Null (базовый тип 7, дополнительное значение 22) становится JSON null.
- Действительное (floating-point) значение (базовый тип 7, дополнительное значение 25 - 27) становится числом JSON, если оно конечно (т. е. представимо числом JSON), а неконечные значения (NaN, положительные и отрицательные Infinity) заменяются подстановочным значением.

- Любое другое простое значение (базовый тип 7, любое дополнительное значение не указанное выше) представляется символом подстановки.
- Большое число (bignum, базовый тип 6, тег 2 или 3) представляется кодированием его строки байтов в base64url без дополнения и становится строкой JSON. Для тега 3 (отрицательное bignum) перед base-кодированным значением помещается символ тильды ~ (преобразование в binary blob вместо числа для предотвращения возможно переполнения в декодере JSON).
- Строка байтов с рекомендацией по кодированию (базовый тип 6, теги 21 - 23) кодируется, как указано в рекомендации, и становится строкой JSON.
- Для всех других тегов (базовый тип 6, любой тег) содержимое тега представляется как значение JSON. А номер тега игнорируется.
- Элементы неопределённого размера до преобразования конвертируются в конкретный размер.

Преобразователь CBOR в JSON может сохранить профиль JSON (I-JSON) [RFC7493] для максимальной совместимости и повышения уверенности в том, что выход JSON можно обработать с предсказуемым результатом. Например, есть влияние на диапазон целых чисел, которые можно представить надёжно, а также на элементы верхнего уровня, которые могут поддерживаться старыми реализациями JSON.

6.2. Преобразование JSON в CBOR

Все значения JSON, будучи закодированными, напрямую отображаются в одно или несколько значений CBOR. Как и при любой генерации CBOR нужно принимать решение относительно представления чисел, как отмечено ниже.

- Числа JSON без дробной части (целые) представляются целыми числами (базовые типы 0 и 1, возможен тип 6, теги 2 и 3) с выбором кратчайшей формы. Числа, длиннее заданного реализацией порога могут представляться действительными (floating-point) значениями. По умолчанию принят диапазон представления от $-2^{53}+1$ до $2^{53}-1$ (полный диапазон целых чисел в представлении binary64, часто применяемом для декодирования JSON [RFC7493]). Протокол на основе CBOR или базовая реализация конвертера может выбрать диапазон $-2^{32}..2^{32}-1$ или $-2^{64}..2^{64}-1$ (полное использование диапазона целых чисел, доступных в CBOR с uint32_t и uint64_t, соответственно) и даже $-2^{31}..2^{31}-1$ или $-2^{63}..2^{63}-1$ (популярные диапазоны целых чисел со знаком и дополнением до 2). Если представление JSON создано реализацией JavaScript, точность уже ограничена значением не более 53 битов.
- Числа с дробной частью представляются действительными (floating-point) значениями с преобразованием двоичных значений в десятичные с точностью, обеспечиваемой форматом IEEE 754 binary64. Математическое значение числа JSON преобразуется в binary64 с использованием процедуры roundTiesToEven из параграфа 4.3.1 в [IEEE754]. Затем при кодировании CBOR применяется предпочтительная сериализация в кратчайшее представления floating-point с сохранением значения, например, число 1,5 преобразуется в 16-битовое значение floating-point (не все реализации способны эффективно определять кратчайшую форму). Вместо принятой по умолчанию точности binary64 может применяться заданная реализацией точность преобразования, влияющая на результат. Десятичные представления следует применять на стороне CBOR лишь при указании этого в протоколе.

Формат CBOR был разработан для обеспечения в общем случае более компактного представления, чем JSON. Одной из возможных стратегий реализации преобразования JSON в CBOR является кодирование на месте в одном буфере. В этом случае нужно внимательно рассмотреть множество патологических случаев, таких как преобразование некоторых строк без экранирования или с очень небольшим числом экранирований и размером больше (возможно, много больше) 255 байтов в строки UTF-8 формата CBOR. Точно так же некоторые двоичные представления floating-point могут возникать из некоторых десятичных представлений (1,1, 1e9) в JSON. Это может быть сложно для исправления и все возникающие уязвимости могут использоваться злоумышленником.

7. Будущее развитие CBOR

Успешные протоколы развиваются с течением времени. Появляются новые идеи, улучшается реализация платформ, сопровождающие протоколы разрабатываются и развиваются, добавляются требования от приложений и протоколов. Таким образом, содействие развитию протокола является важным при разработке любого нового протокола.

Для протоколов, которые будут использовать CBOR, формат CBOR обеспечивает некоторые полезные механизмы, способствующие развитию. Передовой опыт в этом направлении известен, в частности из разработки формата JSON для протоколов на основе JSON. Однако этот опыт выходит за рамки документа.

Тем не менее, развитие самого CBOR вписывается в эти задачи. Формат CBOR разработан для создания стабильной основы разработки протоколов на основе CBOR и возможности развития. Поскольку успешный протокол может развиваться в течение десятилетий, CBOR требуется разрабатывать с учётом такой эволюции. В этом разделе даны некоторые рекомендации по развитию CBOR. Они более субъективны, чем остальная часть этого документа. Кроме того, они не полны, чтобы не превратиться в текст по разработке протоколов.

7.1. Точки расширения

При создании протокола возможности его развития часто включаются в форме точек расширения. Например, может быть пространство кодов, которое не полностью выделено изначально, а протокол разработан так, чтобы допускались реализации, начинающие использовать больше кодов, чем было выделено исходно. Определение размера пространства кодов может быть затруднителеьным, поскольку требуемый диапазон предсказать модет быть сложно. При создании протокола следует следует пытаться сделать пространство кодов достаточно большим, чтобы оно не истощилось в течение предусмотренного срока действия протокола. В CBOR имеется 3 основных точки расширения.

Пространство простых значений (simple, базовый тип 7)

Из 24 эффективных (и 224 менее эффективных) значений, выделено лишь небольшое число. Реализации, получившие неизвестный элемент простых данных могут быть способны легко обработать их как таковые, учитывая простую структуру. Реестр IANA (параграф 9.1) даёт подходящий способ расширения пространства кода.

Пространство тегов (tag, базовый тип 6)

Общее пространство кодов избыточно и распределена лишь очень малая часть. Однако не все коды одинаково эффективны - первые 24 занимают лишь 1 байт (1+0) и половина из них уже распределена. Следующие 232 значения занимают по 2 байта (1+1) и распределены примерно на четверть. Для этих субпространств нужно некоторое наблюдение в течение нескольких следующих десятилетий. Реализации, получившие тег с неизвестным номером, могут обработать вложенное содержимое тега или (предпочтительно) обработать как неизвестный тег, включающий содержимое. Реестр IANA (параграф 9.2) даёт подходящий способ контроля расширения пространства кодов.

Пространство дополнительных значений

Реализация, получившая неизвестное дополнительное значение, не имеет способа продолжить декодирование, поэтому выделение кодов из этого пространства является важным шагом, помимо простого использования в качестве точки расширения. Свободных кодов в этом пространстве осталось мало, см. также параграф 7.2.

7.2. Курирование пространства дополнительных значений

Человеческий разум иногда стремится заполнить небольшие имеющиеся где-то зазоры. Предполагается, что остающиеся пропуски в пространстве кодов будут привлекать своим наличием новые идеи. Данная спецификация не управляет пространством дополнительных значений через реестр IANA. Выделения из этого пространства могут происходить лишь путём обновления этой спецификации.

Для дополнительного значения >= 24 размер дополнительных данных обычно составляет 2^{n-24} байтов. Поэтому дополнительные значения 28 и 29 следует считать кандидатами для 128- и 256-битовых величин, при необходимости добавления их в протокол. Дополнительное значение 30 остаётся единственным доступным для общего распределения и должны быть важные причины для его выделения путём обновления данной спецификации.

8. Диагностическая нотация

CBOR - это двоичный формат обмена. Для облегчения документирования и отладки, в частности, для облегчения взаимодействия между элементами, участвующими в отладке, в этом разделе определятся простая и понятная человеку диагностическая нотация. Фактический обмен по-прежнему выполняется в двоичном формате. Это действительно диагностический формат, он не предназначен для синтаксического анализа, поэтому в документе нет формального определения (как в ABNF). Разработчики, ищущие основанный на тексте формат для представления элементов данных CBOR в файлах конфигурации, могут рассмотреть YAML [YAML].

Диагностическая нотация в общих чертах основана на JSON, как задано RFC 8259, с расширением при необходимости. Нотация следует синтаксису JSON для чисел (целых и действительных), True (true), False (false), Null (null), строк UTF-8, массивов и отображений (отображения в JSON называются объектами, диагностическая нотация расширяет здесь JSON, позволяя использовать в качестве ключа любой элемент данных). Неопределённые значения указываются в виде undefined как в JavaScript. Неконечные действительные числа Infinity, -Infinity и NaN записываются как здесь (этот же способ может применяться в JavaScript, хотя JSON не позволяет этого). Тег записываются как целое число (номер тега), за которым следует содержимое тега в скобках, например, дата в формате, заданном в RFC 3339 (ISO 8601), может быть записана как

```
0("2013-03-21T20:04:00Z")
```

или эквивалентно в относительном формает

```
1(1363896240)
```

Строки байтов указываются в одном из базовых форматов без дополнения в одинарных кавычках с префиксом h для base16, b32 для base32, h32 для base32hex, b64 для base64 или base64url (фактические коды не перекрываются, поэтому строки остаются однозначными). Например, строку байтов 0x12345678 можно записать в виде h'12345678', b32'CI2FM6A', b64'EjRWeA'.

Неназначенные простые значения указываются как "simple()" с подходящим целым числом в скобках. Например "simple(42)" указывает базовый тип 7 со значением 42.

Много полезных расширений для диагностической нотации дано в Приложении G к [RFC8610], Extended Diagnostic Notation (EDN). Нотация может быть расширена в отдельном документе для данных NaN, которые здесь не учтены.

8.1. Индикаторы кодирования

Иногда полезно указать в диагностической нотации, какие из вариантов представления фактически применяются, например, элемент данных, записанный как 1.5 диагностическим декодером может быть закодирован как действительное число половинной, одинарной или двойной точности.

Соглашение для индикаторов кодирования состоит в том, что все, начинающееся с символа _, а последующие символы являются буквами, цифрами или знаком подчёркивания (_) является индикатором кодирования и может игнорироваться не заинтересованной в этих сведениях стороной, например, "_" или "_3". Индикаторы кодирования не обязательны.

Отдельный символ _ может указываться после открывающей квадратной скобки массива для указания того, что элемент данных был представлен в формате неопределённого размера. Например, [_ 1, 2] содержит индикатор неопределённого размера в представлении элемента данных [1, 2]. Симлов _, за которым следует десятичная цифра n, указывает что предшествующий элемент (для массивов и отображений, элемент, начинающийся с предшествующей круглой или квадратной скобки) был закодирован с дополнительным значением 24+n. Например, 1.5_1 - число floating-point половинной точности, а 1.5_3 - двойной. /тот индикатор кодирования не указан в Приложении A. Отметим, что индикатор _ является сокращением формы _7.

Подробная структура строк байтов и текста неопределённого размера может быть указана в форме (_ h'0123', h'4567') и (_ "foo", "bar"). Однако для строк неопределённого размера без блоков внутри (_) будет двусмысленным, в части того, применяется строка байтов (0x5fff) или текста (0x7fff), поэтому такой индикатор не применяется. Взамен могут применяться базовые формы "_ и ""_ и они зарезервированы лишь для отсутствия блоков, а не в качестве кратких форм (разрешено, но бесполезно) кодирования лишь с пустыми блоками, которые нужно обозначать как (_ "), (_ ""), для сохранения структуры блока.

9. Взаимодействие с IANA

Агентство IANA создало два реестра для новых значений CBOR. Реестры раздельны, т. е. не находятся под одним реестром, и следуют правилам [RFC8126]. Агентство IANA также выделило два новых типа носителя, связанную запись CoAP Content-Format и суффикс структурированного синтаксиса.

9.1. Реестр CBOR Simple Values

Агентство IANA создало реестр Concise Binary Object Representation (CBOR) Simple Values [IANA.cbor-simple-values]. Начальные значения приведены в таблице 4. Новые значения из диапазона 0 - 19 выделяются по процедуре Standards Action [RFC8126]. IANA предлагается выделять значения, начиная с 16, чтобы зарезервировать младшие номера для непрерывных блоков (если они есть). Значения из диапазона 32 - 255 выделяются по процедуре Specification Required.

9.2. Реестр CBOR Tags

Агентство IANA создало реестр Concise Binary Object Representation (CBOR) Tags [IANA.cbor-tags]. Теги, заданные в [RFC7049], подробно описаны в параграфе 3.4 и с тех пор уже определены другие теги. Новые значения из диапазона 0 - 23 (1+0) выделяются по процедуре Standards Action, из диапазонов 24 - 255 (1+1) и 256 - 32767 (нижняя половина 1+2) - по процедуре Specification Required. Новые значения от 32768 до 18446744073709551615 (верхняя половина 1+2, 1+4 и 1+8) выделяются по процедуре First Come First Served. Шаблон запроса на регистрацию имеет вид:

- элемент данных;
- семантика (краткая форма).

Кроме того, в запросы First Come First Served следует включать:

- точку контакта;
- описание семантики (URL) - оно не обязательно, URL может указывать что-то Internet-Draft или web-страницы.

Заявителям для диапазонов First Come First Served, предлагающим номер тега, который не представляется 32 битами (больше 4294967295), следует учитывать, что это может препятствовать совместимости с реализациями, которые не поддерживают 64-битовые числа.

9.3. Реестр Media Types

Типом носителя Internet [RFC6838] (MIME type) для одного кодированного элемента данных CBOR является application/cbor, как указано в реестре Media Types [IANA.media-types].

Type name: application
Subtype name: cbor
Required parameters: n/a
Optional parameters: n/a
Encoding considerations: Binary
Security considerations: See Section 10 of RFC 8949.
Interoperability considerations: n/a
Published specification: RFC 8949
Applications that use this media type: Many
Additional information:
Magic number(s): n/a
File extension(s): .cbor
Macintosh file type code(s): n/a
Person & email address to contact for further information: IETF CBOR Working Group (cbor@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author: IETF CBOR Working Group (cbor@ietf.org)
Change controller: The IESG (iesg@ietf.org)

9.4. Реестр CoAP Content-Format

CoAP Content-Format для CBOR зарегистрирован в субреестре CoAP Content-Formats реестра Constrained RESTful Environments (CoRE) Parameters [IANA.core-parameters]:

Media Type: application/cbor
Encoding: -
ID: 60
Reference: RFC 8949

9.5. Реестр Structured Syntax Suffix

Суффиксом структурированного синтаксиса [RFC6838] для типов носителей, основанных на одном кодированном элементе данных CBOR, служит +cbor, зарегистрированный в реестре Structured Syntax Suffixes [IANA.structured-suffix]

Name: Concise Binary Object Representation (CBOR)
+suffix: +cbor
References: RFC 8949
Encoding Considerations: CBOR is a binary format.
Interoperability Considerations: n/a
Fragment Identifier Considerations: The syntax and semantics of fragment identifiers specified for +cbor SHOULD be as specified for "application/cbor". (At publication of RFC 8949, there is no fragment identification syntax defined for "application/cbor".)
The syntax and semantics for fragment identifiers for a specific "xxx/yyy+cbor" SHOULD be processed as follows:

- For cases defined in +cbor, where the fragment identifier resolves per the +cbor rules, then process as specified in +cbor.
- For cases defined in +cbor, where the fragment identifier does not resolve per the +cbor rules, then process as specified in "xxx/yyy+cbor".
- For cases not defined in +cbor, then process as specified in "xxx/yyy+cbor".

Security Considerations: See Section 10 of RFC 8949.
Contact: IETF CBOR Working Group (cbor@ietf.org) or IETF Applications and Real-Time Area (art@ietf.org)
Author/Change Controller: IETF

10. Вопросы безопасности

Сетевое приложение может иметь уязвимости в логике обработки входящих данных. Сложные синтаксические анализаторы хорошо известны как вероятный источник таких уязвимостей, как возможность удаленно вызвать отказ узла или даже выполнить на нем произвольный код. CBOR пытается сократить возможности внедрения таких уязвимостей путём снижения сложности синтаксических анализаторов за счёт придания смысла всему диапазону кодированных значений, когда это возможно.

Поскольку декодеры CBOR часто используются в качестве первого этапа обработки непроверенных входных данных, они должны быть полностью готовы ко всем типам враждебных данных, которые могут быть созданы для повреждения, переполнения или захвата управления системой, декодирующей элементы данных CBOR. Декодер CBOR должен предполагать, что весь ввод может быть враждебным, даже если данные проверены межсетевым экраном, поступили по защищённому каналу, такому как TLS, зашифрованы или подписаны, поступили из считающегося доверенным источника.

В параграфе 4.1 даны примеры ограничений функциональной совместимости при использовании ограниченного декодера CBOR вместе с кодером CBOR, использующим непредпочтительную сериализацию. Когда один элемент данных воспринимают декодер с ограничениями и полный декодер, могут возникать проблемы безопасности, которыми атакующий может воспользоваться для внедрения или манипуляций содержимым.

Как уже обсуждалось в этом документе, есть много значений, которые могут считаться «эквивалентными» в одних обстоятельствах и разными - в других. Примером может служить численное значение «один», которое можно представить целым числом или bignum. Система, обрабатывающая ввод CBOR может воспринять или отвергнуть любую из этих форм (или обе). Это будет влиять на безопасность программы, использующей входные данные.

Враждебный ввод может служить для переполнения буферов, переполнения или потери значимости (underflow) в целочисленной арифметике, а также вызывать иные нарушения при декодировании. Элементы данных CBOR могут иметь размеры, которые преднамеренно сделаны слишком большими или слишком малыми. Атаки с истощением ресурсов могут пытаться вынудить декодер к выделению очень больших элементов данных (строк, массивов, отображений или даже чисел с произвольной точностью) или превышению глубины стека за счёт многоуровневой вложенности. Декодеры должны включать соответствующее управления ресурсами для смягчения таких атак. Элементы очень большого размера могут также служить для попыток использовать переполнение целых чисел.

Декодер CBOR по определению воспринимает лишь корректно сформированные данные CBOR - это первый шаг к отказоустойчивости. Получение некорректно сформированных данных CBOR не ведёт к их дальнейшей обработке после обнаружения некорректности. По возможности любые данные, которые уже декодированы к этому моменту, не должны оказывать влияния на приложение, использующее декодер CBOR.

В дополнение к проверке корректности декодер CBOR может проверять пригодность (validity) данных CBOR или оставить проверку пригодности приложению, использующему декодер. Этот выбор должен чётко указываться в документации декодера. Сверх проверки пригодности на уровне CBOR приложению нужно удостовериться в соответствии ввода приклатному протоколу, сериализуемому в CBOR.

Проверка ввода сама может потреблять ресурсы и обычно они линейно зависят от размера ввода, а это означает, что атакующему приходится затрачивать ресурсы, сопоставимые с затрачиваемыми на проверку ввода. Однако у атакующего может быть возможность создать входные данные, обработка которых займёт больше времени, чем их подготовка злоумышленником. Обработка чисел с произвольной точностью может усилить роста расхода ресурсов. Кроме того, некоторые реализации хэш-таблиц, используемые декодерами для представления отображений в памяти, могут быть атакованы с целью вызвать квадратичный рост расхода ресурсов, если не реализованы секретные ключи (см. раздел 7 в [SIPHASH_LNCS], а также [SIPHASH_OPEN]) или иные меры смягчения. Злоумышленник может воспользоваться такой нелинейностью для истощения ресурсов при входной проверке или до неё, поэтому следует избегать этого в реализации декодеров CBOR. Отметим, что определения номеров тегов и их реализация могут создавать проблемы безопасности этого типа, поэтому следует рассматривать соображения безопасности при определении тегов.

Кодеры CBOR не получают входные данные напрямую из сети и поэтому не подвержены таким же атакам, как декодеры CBOR. Однако кодеры CBOR часто включают API, принимающие данные от другого уровня реализации, и могут быть атакованы через эти API. Устройство и реализация таких API должны учитывать, что поведение вызывающей стороны может быть враждебным или основанным на ошибках кода. Следует проверять ввод на предмет переполнения буферов, переполнения и потери значимости в целочисленной арифметике, а также наличие иных ошибок, способных нарушить работу кодера.

Протоколы нужно задавать так, чтобы возможные множественные интерпретации надёжно сводились в одну. Например, злоумышленник может использовать непригодные входные данные, такие как дубликаты ключей в отображении, или воспользоваться разной точностью при обработке чисел, чтобы вынудить одно приложение к интерпретации, отличающейся от принятой другим. Для облегчения согласованной интерпретации реализациям кодеров и декодеров следует поддерживать режим проверки пригодности (параграф 5.4). Отметим однако, что базовый декодер не может знать всех требований приложения к входным данным, в результате чего приложение должно выполнять свою проверку ввода. Поскольку набор заданных тегов меняется, приложение может использовать номер тега, для которого ещё не поддерживается проверка пригодности в применяемом базовом декодере. Поэтому в

документации базовых декодеров нужно указывать поддерживаемые теги и выполняемые проверки номеров тегов и базовые проверки CBOR (UTF-8, дубликаты ключей отображения).

В параграфе 3.4.3 отмечено, что использование непредпочтительного выбора для представления bignum вместо базового целого числа при кодировании не предназначено быть семантикой приложения, но может стать таковой, если принимающее данные CBOR приложение использует декодер в базовой модели данных. Это несоответствие создаёт проблему безопасности, если два варианта семантики различаются. Таким образом, использующим CBOR приложениям нужно указывать модель данных, которую они применяют при каждом использовании данных CBOR.

Данные CBOR часто преобразуются в иные форматы. Во многих случаях CBOR имеет более выразительные типы по сравнению с другими форматами и это особенно ярко проявляется для JSON. Потеря сведений о типе может вызывать проблемы безопасности для систем, обрабатывающих менее выразительные данные.

В параграфе 6.2 описан возможный вариант применения преобразования между CBOR и JSON, который может быть атакован, если злоумышленнику известно выполняющее преобразование приложение.

Соображения безопасности при использовании base16 и base64 из [RFC4648] и UTF-8 из [RFC3629] применимы и к CBOR.

11. Литература

11.1. Нормативные документы

[C] International Organization for Standardization, "Information technology - Programming languages — C", Fourth Edition, ISO/IEC 9899:2018, June 2018, <https://www.iso.org/standard/74528.html>.

[Cplusplus20] International Organization for Standardization, "Programming languages - C++", Sixth Edition, ISO/IEC DIS 14882, ISO/IEC ISO/IEC JTC1 SC22 WG21 N 4860, March 2020, <https://isocpp.org/files/papers/N4860.pdf>.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229, <https://ieeexplore.ieee.org/document/8766229>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[TIME_T] The Open Group, "The Open Group Base Specifications", Section 4.16, 'Seconds Since the Epoch', Issue 7, 2018 Edition, IEEE Std 1003.1, 2018, <https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16>.

11.2. Дополнительная литература

[ASN.1] International Telecommunication Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 2015, <https://www.itu.int/rec/T-REC-X.690-201508-I/en>.

[BSON] Various, "BSON - Binary JSON", <http://bsonspec.org/>.

[CBOR-TAGS] Bormann, C., "Notable CBOR Tags", Work in Progress, Internet-Draft, draft-bormann-cbor-notable-tags-02, 25 June 2020, <https://tools.ietf.org/html/draft-bormann-cbor-notable-tags-02>.

[ECMA262] Ecma International, "ECMAScript 2020 Language Specification", Standard ECMA-262, 11th Edition, June 2020, <https://www.ecma-international.org/publications/standards/Ecma-262.htm>.

[Err3764] RFC Errata, Erratum ID 3764, RFC 7049, <https://www.rfc-editor.org/errata/eid3764>.

[Err3770] RFC Errata, Erratum ID 3770, RFC 7049, <https://www.rfc-editor.org/errata/eid3770>.

[Err4294] RFC Errata, Erratum ID 4294, RFC 7049, <https://www.rfc-editor.org/errata/eid4294>.

[Err4409] RFC Errata, Erratum ID 4409, RFC 7049, <https://www.rfc-editor.org/errata/eid4409>.

[Err4963] RFC Errata, Erratum ID 4963, RFC 7049, <https://www.rfc-editor.org/errata/eid4963>.

[Err4964] RFC Errata, Erratum ID 4964, RFC 7049, <https://www.rfc-editor.org/errata/eid4964>.

[Err5434] RFC Errata, Erratum ID 5434, RFC 7049, <https://www.rfc-editor.org/errata/eid5434>.

[Err5763] RFC Errata, Erratum ID 5763, RFC 7049, <https://www.rfc-editor.org/errata/eid5763>.

[Err5917] RFC Errata, Erratum ID 5917, RFC 7049, <https://www.rfc-editor.org/errata/eid5917>.

[IANA.cbor-simple-values] IANA, "Concise Binary Object Representation (CBOR) Simple Values", <https://www.iana.org/assignments/cbor-simple-values>.

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA.structured-suffix] IANA, "Structured Syntax Suffixes", <https://www.iana.org/assignments/media-type-structured-suffix>.

[MessagePack] Furuhashi, S., "MessagePack", <https://msgpack.org/>.

[PCRE] Hazel, P., "PCRE - Perl Compatible Regular Expressions", <https://www.pcre.org/>.

[RFC0713] Haverty, J., "MSDTP-Message Services Data Transmission Protocol", RFC 713, DOI 10.17487/RFC0713, April 1976, <https://www.rfc-editor.org/info/rfc713>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <https://www.rfc-editor.org/info/rfc7493>.

[RFC7991] Hoffman, P., "The "xml2rfc" Version 3 Vocabulary", RFC 7991, DOI 10.17487/RFC7991, December 2016, <https://www.rfc-editor.org/info/rfc7991>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8618] Dickinson, J., Hague, J., Dickinson, S., Manderson, T., and J. Bond, "Compacted-DNS (C-DNS): A Format for DNS Packet Capture", RFC 8618, DOI 10.17487/RFC8618, September 2019, <https://www.rfc-editor.org/info/rfc8618>.

[RFC8742] Bormann, C., "Concise Binary Object Representation (CBOR) Sequences", RFC 8742, DOI 10.17487/RFC8742, February 2020, <https://www.rfc-editor.org/info/rfc8742>.

[RFC8746] Bormann, C., Ed., "Concise Binary Object Representation (CBOR) Tags for Typed Arrays", RFC 8746, DOI 10.17487/RFC8746, February 2020, <https://www.rfc-editor.org/info/rfc8746>.

[SIPHASH_LNCS] Aumasson, J. and D. Bernstein, "SipHash: A Fast Short-Input PRF", Progress in Cryptology - INDOCRYPT 2012, pp. 489-508, DOI 10.1007/978-3-642-34931-7_28, 2012, <https://doi.org/10.1007/978-3-642-34931-7_28>.

[SIPHASH_OPEN] Aumasson, J. and D.J. Bernstein, "SipHash: a fast short-input PRF", <https://www.aumasson.jp/siphash/siphash.pdf>.

[YAML] Ben-Kiki, O., Evans, C., and I.d. Net, "YAML Ain't Markup Language (YAML[TM]) Version 1.2", 3rd Edition, October 2009, <https://www.yaml.org/spec/1.2/spec.html>.

Приложение A. Примеры кодированных элементов данных CBOR

В таблице 6 представлено кодирование CBOR для некоторых значений в шестнадцатеричной форме (справа) и диагностическая нотация для них (слева). Отметим, что строка "\u00fc" является одной из форм диагностической нотации для строк UTF-8 с одним символом Unicode U+00FC (LATIN SMALL LETTER U WITH DIAERESIS, "ü"), "\u6c34" - строки UTF-8 с одним символом U+6C34 (CJK UNIFIED IDEOGRAPH-6C34, "水"), часто представляющим воду (water), а "\ud800\udd51" - строки UTF-8 с одним символом U+10151 (GREEK ACROPHONIC ATTIC FIFTY STATERS, "𐅑"). Отметим, что все эти односимвольные строки можно представить в обычно диагностической нотации UTF-8, если не требуется применять лишь символы ASCII. В диагностической нотации для bignums предполагаемое численное значение указывается в десятичной форме (например, 18446744073709551616), а не строкой байтов с тегом (например, 2(h'010000000000000000')).

Таблица 6. Примеры кодированных элементов данных CBOR.

Диагностика	***Кодирование***
0	0x00
1	0x01

10	0x0a
23	0x17
24	0x1818
25	0x1819
100	0x1864
1000	0x1903e8
1000000	0x1a000f4240
1000000000000	0x1b000000e8d4a51000
18446744073709551615	0x1bffffffffffffff
18446744073709551616	0xc249010000000000000
-18446744073709551616	0x3bffffffffffffff
-18446744073709551617	0xc349010000000000000
-1	0x20
-10	0x29
-100	0x3863
-1000	0x3903e7
0.0	0xf90000
-0.0	0xf98000
1.0	0xf93c00
1.1	0xfb3ff199999999999a
1.5	0xf93e00
65504.0	0xf97bff
100000.0	0xfa47c35000
3.4028234663852886e+38	0xfa7f7fffff
1.0e+300	0xfb7e37e43c8800759c
5.960464477539063e-8	0xf90001
0.00006103515625	0xf90400
-4.0	0xf9c400
-4.1	0xfbc010666666666666
Infinity	0xf97c00
NaN	0xf97e00
-Infinity	0xf9fc00
Infinity	0xfa7f800000
NaN	0xfa7fc00000
-Infinity	0xfaff800000
Infinity	0xfb7ff0000000000000
NaN	0xfb7ff8000000000000
-Infinity	0xfbfff0000000000000
false	0xf4
true	0xf5
null	0xf6
undefined	0xf7
simple(16)	0xf0
simple(255)	0xf8ff
0("2013-03-21T20:04:00Z")	0xc074323031332d30332d32315432303a30343a30305a
1(1363896240)	0xc11a514b67b0
1(1363896240.5)	0xc1fb41d452d9ec200000
23(h'01020304')	0xd74401020304
24(h'6449455446')	0xd818456449455446
32("http://www.example.com")	0xd82076687474703a2f2f7777772e6578616d706c652e636f6d
h''	0x40
h'01020304'	0x4401020304
""	0x60
"a"	0x6161
"IETF"	0x6449455446
"\"\\"	0x62225c
"\u00fc"	0x62c3bc
"\u6c34"	0x63e6b0b4
"\ud800\udd51"	0x64f0908591
[]	0x80
[1, 2, 3]	0x83010203
[1, [2, 3], [4, 5]]	0x8301820203820405
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]	0x98190102030405060708090a0b0c0d0e0f101112131415161718181819
{}	0xa0
{1: 2, 3: 4}	0xa201020304
{"a": 1, "b": [2, 3]}	0xa26161016162820203
["a", {"b": "c"}]	0x826161a161626163
{"a": "A", "b": "B", "c": "C", "d": "D", "e": "E"}	0xa5616161416162614261636143616461446165614 5
(_ h'0102', h'030405')	0x5f42010243030405ff
(_ "strea", "ming")	0x7f657374726561646d696e67ff
[_]	0x9fff
[_ 1, [2, 3], [_ 4, 5]]	0x9f018202039f0405ffff
[_ 1, [2, 3], [4, 5]]	0x9f01820203820405ff
[1, [2, 3], [_ 4, 5]]	0x83018202039f0405ff
[1, [_ 2, 3], [4, 5]]	0x83019f0203ff820405

[_ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]	0x9f0102030405060708090a0b0c0d0e0f10111213141516171 8181819ff
{_ "a": 1, "b": [_ 2, 3]}	0xbf61610161629f0203ffff
["a", {_ "b": "c"}]	0x826161bf61626163ff
{_ "Fun": true, "Amt": -2}	0xbf6346756ef563416d7421ff

Приложение B. Таблица переходов для начального байта

Для краткости в таблице 7 не представлены начальные байты, зарезервированные для будущих расширений, и включена лишь подборка начальных байтов, которые могут служить для необязательных функций. Все целые числа без знака указаны в сетевом порядке байтов.

Таблица 7. Таблица перехода для начального байта.

Байт	***Структура и семантика***
0x00..0x17	целое число без знака 0x00..0x17 (0..23)
0x18	целое число без знака, затем 1 байт uint8_t
0x19	целое число без знака, затем 2 байта uint16_t
0x1a	целое число без знака, затем 4 байта uint32_t
0x1b	целое число без знака, затем 8 байтов uint64_t
0x20..0x37	отрицательное целое число -1-0x00..-1-0x17 (-1..-24)
0x38	отрицательное целое число -1-n, затем 1 байт uint8_t для n
0x39	отрицательное целое число -1-n, затем 2 байта uint16_t для n
0x3a	отрицательное целое число -1-n, затем 4 байта uint32_t для n
0x3b	отрицательное целое число -1-n, затем 8 байтов uint64_t для n
0x40..0x57	строка байтов, затем 0x00..0x17 байтов
0x58	строка байтов, затем 1 байт uint8_t для n и n байтов
0x59	строка байтов, затем 2 байта uint16_t для n и n байтов
0x5a	строка байтов, затем 4 байта uint32_t для n и n байтов
0x5b	строка байтов, затем 8 байтов uint64_t для n и n байтов
0x5f	строка байтов, затем строка байтов и break
0x60..0x77	строка UTF-8, затем 0x00..0x17 байтов
0x78	строка UTF-8, затем 1 байт uint8_t для n и n байтов
0x79	строка UTF-8, затем 2 байта uint16_t для n и n байтов
0x7a	строка UTF-8, затем 4 байта uint32_t для n и n байтов
0x7b	строка UTF-8, затем 8 байтов uint64_t для n и n байтов
0x7f	строка UTF-8, затем строка байтов и break
0x80..0x97	массив, затем 0x00..0x17 элементов данных
0x98	массив, затем 1 байт uint8_t для n и n элементов данных
0x99	массив, затем 2 байта uint16_t для n и n элементов данных
0x9a	массив, затем 4 байта uint32_t для n и n элементов данных
0x9b	массив, затем 8 байтов uint64_t для n и n элементов данных
0x9f	массив, затем элементы данных и break
0xa0..0xb7	отображение, затем 0x00..0x17 пар элементов данных
0xb8	отображение, затем 1 байт uint8_t для n и n пар элементов данных
0xb9	отображение, затем 2 байта uint16_t для n и n пар элементов данных
0xba	отображение, затем 4 байта uint32_t для n и n пар элементов данных
0xbb	отображение, затем 8 байтов uint64_t для n и n пар элементов данных
0xbf	отображение, затем пары элементов данных и break
0xc0	текстовое представление даты-времени, затем элемент данных (параграф 3.4.1)
0xc1	дата и время на основе эпохи, затем элемент данных (параграф 3.4.2)
0xc2	bignum без знака, затем строка байтов
0xc3	отрицательное значение bignum, затем строка байтов
0xc4	десятичная дробь, затем массив (параграф 3.4.4)
0xc5	bigfloat, затем массив (параграф 3.4.4)
0xc6..0xd4	(тег)
0xd5..0xd7	ожидаемое преобразование, затем элемент данных (параграф 3.4.5.2)
0xd8..0xdb	(дополнительные теги - 1/2/4/8 номера тега, затем элемент данных)
0xe0..0xf3	(простое значение)
0xf4	false
0xf5	true
0xf6	null
0xf7	undefined
0xf8	(простое значение, затем 1 байт)
0xf9	действительное число половинной точности (2 байта IEEE 754)
0xfa	действительное число одинарной точности (4 байта IEEE 754)
0xfb	действительное число двойной точности (8 байтов IEEE 754)
0xff	код завершения break

Приложение C. Псевдокод

Корректность формы элемента CBOR можно проверить с помощью псевдокода, представленного на рисунке 1. Данные считаются корректно сформированными лишь при выполнении обоих указанных ниже условий.

- Результат выполнения псевдокода отличается от fail.
- После выполнения псевдокода не остаётся байтов на входе (за исключением потоковых приложений).

Псевдокод предполагает выполнение указанных ниже условий.

- Функция take(n) считывает n байтов входных данных и возвращает их в форме строки байтов. Если n байтов больше не доступно, take(n) возвращает отказ (fail).

- Функция uint() преобразует строку байтов в целое число без знака, интерпретируя строку в сетевом порядке байтов.
- Арифметические операции выполняются как в языке C.
- Все переменные являются целыми числами без знака с достаточным размером.

Отметим, что well_formed возвращает базовый тип для корректно сформированных элементов определённого размера и значение 99 для элементов неопределённого размера (или -1 для кода завершения break, если установлен флаг breakable). Это используется в well_formed_indefinite, для проверки того, что строки неопределённого размера состоят лишь из блоков определённого размера.

```
well_formed(breakable = false) {
  // обработка начальных байтов
  ib = uint(take(1));
  mt = ib >> 5;
  val = ai = ib & 0x1f;
  switch (ai) {
    case 24: val = uint(take(1)); break;
    case 25: val = uint(take(2)); break;
    case 26: val = uint(take(4)); break;
    case 27: val = uint(take(8)); break;
    case 28: case 29: case 30: fail();
    case 31:
      return well_formed_indefinite(mt, breakable);
  }
  // обработка содержимого
  switch (mt) {
    // в случаях 0, 1, 7 содержимого нет и просто используется val
    case 2: case 3: take(val); break; // байты/UTF-8
    case 4: for (i = 0; i < val; i++) well_formed(); break;
    case 5: for (i = 0; i < val*2; i++) well_formed(); break;
    case 6: well_formed(); break;     // 1 вложенный элемент
    case 7: if (ai == 24 && val < 32) fail(); // плохое простое значение
  }
  return mt;                    // элемент данных определённого размера
}

well_formed_indefinite(mt, breakable) {
  switch (mt) {
    case 2: case 3:
      while ((it = well_formed(true)) != -1)
        if (it != mt)           // нужен блок определённого размера
          fail();               // с тем же типом
      break;
    case 4: while (well_formed(true) != -1); break;
    case 5: while (well_formed(true) != -1) well_formed(); break;
    case 7:
      if (breakable)
        return -1;              // указывает прерывание (break out)
      else fail();              // нет вложенных элементов
                                // неопределённого размера
    default: fail();            // неверное значение mt
  }
  return 99;                    // элемент неопределённого размера
}
```

Рисунок 1. Псевдокод для проверки пригодности формы.

Отметим, что оставшаяся сложность полного декодера CBOR состоит в представлении декодированных данных приложению в подходящей форме.

Базовые типы 0 и 1 можно закодировать в C из целого числа со знаком, фактически без if-then-else для знака (Рисунок 2). используется то, что (-1-n) - преобразование для базового типа 1 - совпадает с ~n (побитовое дополнение) в арифметике C, ~n можно выразить как $(-1)^n$ для отрицательного числа, тогда как 0^n оставляет n без изменения в случае неотрицательного. Знак числа можно представить как -1 для отрицательного и 0 в ином случае (0 или положительное число) путём арифметического сдвига числа на 1 бит меньше его размера (например, на 63 для 64-битового числа).

```
void encode_sint(int64_t n) {
  uint64t ui = n >> 63;    // извлечение знака
  unsigned mt = ui & 0x20; // извлечение базового типа
  ui ^= n;                 // дополнение отрицательного числа
  if (ui < 24)
    *p++ = mt + ui;
  else if (ui < 256) {
    *p++ = mt + 24;
    *p++ = ui;
  } else
      ...
```

Рисунок 2. Псевдокод для кодирования целого числа со знаком.

В параграфе 1.2 указаны некоторые конкретные допущения о профиле языка C, применяемого в этих фрагментах кода.

Приложение D. Половинная точность

Поскольку действительные числа (floating-point) с половинной точностью были добавлены в IEEE 754 лишь в 2008 г. [IEEE754], сегодняшние программные платформы зачастую имеют для них лишь ограниченную поддержку. Для этих чисел очень просто включить хотя бы поддержку декодирования. Пример небольшого декодера на языке C для

действительных чисел с половинной точностью приведён на рисунке 3, а похожей программы Python - на рисунке 4. Этот код предполагает, что 2-байтовое значение уже декодировано как (короткое без знака) целое число с сетевым порядком байтов (как делает псевдокод из Приложения C).

```
#include <math.h>

double decode_half(unsigned char *halfp) {
  unsigned half = (halfp[0] << 8) + halfp[1];
  unsigned exp = (half >> 10) & 0x1f;
  unsigned mant = half & 0x3ff;
  double val;
  if (exp == 0) val = ldexp(mant, -24);
  else if (exp != 31) val = ldexp(mant + 1024, exp - 25);
  else val = mant == 0 ? INFINITY : NAN;
  return half & 0x8000 ? -val : val;
}
```

Рисунок 3. Код C для декодера с половинной точностью.

```
import struct
from math import ldexp

def decode_single(single):
    return struct.unpack("!f", struct.pack("!I", single))[0]

def decode_half(half):
    valu = (half & 0x7fff) << 13 | (half & 0x8000) << 16
    if ((half & 0x7c00) != 0x7c00):
        return ldexp(decode_single(valu), 112)
    return decode_single(valu | 0x7f800000)
```

Рисунок 4. Код Python для декодера с половинной точностью.

Приложение E. Сравнение с другими двоичными форматами

Предложение CBOR следует за чередой двоичных форматов, длинной как история компьютеров. Разные форматы создавались с различными целями. В большинстве случаев цели формата не объявлялись, хотя они иногда подразумеваются контекстом, где формат изначально применялся. Некоторые форматы предназначались для универсального применения, хотя история показала, что ни один формат не отвечает потребностям всех протоколов и приложений.

CBOR отличается от многих из этих форматов тем, что он чётко задаёт набор целей и пытается достичь их. В этом приложении несколько десятков форматов сравнивается с целями CBOR, чтобы помочь читателю решить вопрос о применении CBOR или иного формата для конкретного протокола или приложения.

Отметим, что обсуждение не нацелено на критику того или иного формата. Насколько известно авторам, ни один из форматов до CBOR не предназначался для объявленных CBOR целей с учётом заданных для них приоритетов. Краткий список целей из параграфа 1.1 приведён ниже.

1. Однозначное кодирование наиболее распространённых форматов из стандартов Internet.
2. Компактность кодеров и декодеров.
3. Отсутствие требования описания схемы.
4. Достаточно компактная сериализация.
5. Применимость в приложениях с ограничениями и без таковых.
6. Хорошая преобразуемость в JSON.
7. Расширяемость.

Обсуждение CBOR и других форматов в плане иного набора целей представлено в разделе 5 и Приложении C [RFC8618].

E.1. ASN.1 DER, BER, PER

[ASN.1] имеет много вариантов сериализации. В IETF чаще всего применяются DER и BER. Последовательный вывод недостаточно компактен для многих элементов, а код для декодирования числовых элементов достаточно сложен для устройств с ограничениями.

Немногие (если есть) протоколы IETF приняли один из нескольких вариантов правил кодирования пакетов (Packed Encoding Rules или PER). Для этого есть много причин и одна из них как правило состоит в том, что PER использует схему даже для анализа поверхностной структуры элемента данных, требуя значительной инструментальной поддержки. Имеются разные версии языка схем ASN.1, что также препятствует внедрению.

E.2. MessagePack

[MessagePack] - это компактный, широко распространенный формат счетной двоичной сериализации, во многом походий на CBOR, хотя и менее привычный. Модель данных может служить для представления данных JSON, а также MessagePack применяется во многих приложениях с удаленным вызовом процедур (remote procedure call или RPC) а также для длительного хранения данных. Формат MessagePack практически стабилен с момента публикации около 2011 г., новых версий ещё нет. Развитие сдерживает необходимость поддержки полной совместимости с уже хранящимися данными и доступность для расширения лишь нескольких байт-кодов. Запросы сообщества пользователей MessagePack в течение многих лет на разделение двоичных и текстовых строк недавно привели к расширению, которое будет оставлять необработанные (raw) данные MessagePack неоднозначными при использовании для двоичных и текстовых данных. Механизм расширения MessagePack остаётся непонятным.

E.3. BSON

[BSON] - формат данных, разработанный для хранения отображений в стиле JSON (объекты JSON) в базе данных MongoDB. Его основным отличием является возможность обновления на месте, что препятствует компактности. BSON применяет счетное представление за исключением ключей отображений, которые используют null-байт в конце. Хотя BSON можно применять для представления подобных JSON объектов в линии, в спецификации преобладают требования баз данных и формат стал несколько причудливым. Статус способов расширения BSON остаётся неясным.

E.4. MSDTP - RFC 713

MSDTP (Message Services Data Transmission) является очень ранним примером компактного формата сообщений, описанным [RFC0713] в 1976 г. Формат включён изза исторической ценности, а не по причине широкого применения.

E.5. Лаконичность в линии

Хотя цель CBOR в части компактности кодеров и декодеров является более приоритетной, чем обеспечение лаконичности в линии, многих интересует компактность передачи. В таблице 8 приведены некоторые примеры кодирования для простого вложенного массива [1, [2, 3]], где поддерживается некая форма кодирования неопределённого размера, а также показано кодирование при неопределенном размере внешнего массива [_ 1, [2, 3]].

Таблица 8. Примеры разных уровней лаконичности.

Формат	***[1, [2, 3]]***	***[_ 1, [2, 3]]***
RFC 713	c2 05 81 c2 02 82 83	
ASN.1 BER	30 0b 02 01 01 30 06 02 01 02 02 01 03	30 80 02 01 01 30 06 02 01 02 02 01 03 00 00
MessagePack	92 01 92 02 03	
BSON	22 00 00 00 10 30 00 01 00 00 00 04 31 00 13 00 00 00 10 30 00 02 00 00 00 10 31 00 03 00 00 00 00 00	
CBOR	82 01 82 02 03	9f 01 82 02 03 ff

Приложение F. Некорректные формы и примеры

При декодировании элементов данных CBOR могут возникать три основных ошибки формы элемента.

Слишком много данных
: При вводе остались неиспользованные байты. Это является ошибкой лишь в том случае, когда приложение считало, что входные байты охватывают ровно один элемент данных. Если приложение использует присущее CBOR саморазграничение, чтобы разрешить дополнительные данные после элемента данных, как это делается, например, в последовательностях CBOR [RFC8742], декодер CBOR может просто указать, какая часть ввода не была использована.

Слишком мало данных
: Доступных при вводе данных недостаточно для полного элемента данных CBOR. Это может указывать отсечку ввода или попытку декодировать случайные данные как CBOR. Однако для некоторых приложений это может не быть ошибкой, поскольку приложение может быть не уверено в получении всех данных или ждать поступления на вход дополнительных байтов. Некоторые из таких приложений могут иметь верхнее ограничение числа возможных дополнительных данных и декодер здесь может указать, что кодированный элемент данных CBOR не может поместиться с учётом этого предела.

Синтаксическая ошибка
: Входные данные не соответствуют требованиям кодирования CBOR и это не исправить добавлением (или удалением) данных в конце.

Ошибки первого типа рассматриваются в первом абзаце (и списке - не осталось байтов) Приложения C, а ошибки второго типа - во втором абзаце и списке этого приложения. Третий тип ошибок идентифицируется в псевдокоде конкретными экземплярами вызова fail() в указанном ниже порядке.

- Дополнительное значение из числа зарезервированных (28, 29, 30).
- Базовый тип 7, дополнительное значение 24, значение < 32 (некорректно)
- Некорректная структура строки байтов или текста с неопределённым размером (может включать лишь строки того же базового типа с определенным размером).
- Код завершения break (базовый тип 7, дополнительное значение 31) в позиции значения в отображении, не находящейся в элементе неопределённого размера, куда может быть вложен другой элемент данных.
- Дополнительное значение 31 используется с базовым типом 0, 1 или 6.

F.1. Примеры элементов данных CBOR не пригодных по форме

В этом параграфе приведено несколько примеров некорректных по форме элементов данных CBOR в виде последовательности байтов в шестнадцатеричной форме. Отдельные примеры разделены запятыми.

Примеры ошибок типа 1 (много данных) легко создать добавлением байтов в конец корректного элемента CBOR. Примеры ошибок типа 2 (мало данных) легко создать путём отсечки корректных элементов CBOR. В тестовых наборах может быть полезной специальная проверка неполных элементов данных, которые требуют для завершения много данных (например, путём запуска кодирования очень большой строки).

Преждевременное завершение ввода может произойти в голове или внутри вложенных данных, которые могут быть обычными строками или вложенными элементами (подсчитываются или завершаются кодом break).

Конец ввода в голове
: 18, 19, 1a, 1b, 19 01, 1a 01 02, 1b 01 02 03 04 05 06 07, 38, 58, 78, 98, 9a 01 ff 00, b8, d8, f8, f9 00, fa 00 00, fb 00 00 00

Строка определённого размера с нехваткой данных
: 41, 61, 5a ff ff ff ff 00, 5b ff ff ff ff ff ff ff ff 01 02 03, 7a ff ff ff ff 00, 7b 7f ff ff ff ff ff ff ff 01 02 03

Отображения и массивы определённого размера с нехваткой элементов
: 81, 81 81 81 81 81 81 81 81 81, 82 00, a1, a2 01 02, a1 00, a2 00 00 00

Номер тега без содержимого тега
c0
Строка неопределённого размера без кода завершения break
5f 41 00, 7f 61 00
Отображение или массив неопределённого размера без кода завершения break
9f, 9f 01 02, bf, bf 01 02 01 02, 81 9f, 9f 80 00, 9f 9f 9f 9f ff ff ff ff, 9f 81 9f 81 9f 9f ff ff ff
Ниже представлены примеры 5 подтипов синтаксических ошибок (тип 3).

Подтип 1
Резервные дополнительные значения
1c, 1d, 1e, 3c, 3d, 3e, 5c, 5d, 5e, 7c, 7d, 7e, 9c, 9d, 9e, bc, bd, be, dc, dd, de, fc, fd, fe,
Подтип 2
Резервное двухбайтовое кодирование простых значений
f8 00, f8 01, f8 18, f8 1f
Подтип 3
Блоки некорректного типа в строке неопределённого размера
5f 00 ff, 5f 21 ff, 5f 61 00 ff, 5f 80 ff, 5f a0 ff, 5f c0 00 ff, 5f e0 ff, 7f 41 00 ff
Блоки неопределённого размера в строке неопределённого размера
5f 5f 41 00 ff ff, 7f 7f 61 00 ff ff
Подтип 4
Код завершения вне элемента неопределённого размера
ff
Код завершения в массиве или отображении с определенным размером
81 ff, 82 00 ff, a1 ff, a1 ff 00, a1 00 ff, a2 00 00 ff, 9f 81 ff, 9f 82 9f 81 9f 9f ff ff ff ff
Код завершения в отображении неопределённого размера, задающие нечётное число элементов (break вместо значения)
bf 00 ff, bf 00 00 00 ff
Подтип 5
Базовый тип 0, 1, 6 с дополнительным значением 31
1f, 3f, df

Приложение G. Отличия от RFC 7049

Как указано во введении, этот документ формально отменяет RFC 7049, сохраняя полную совместимость с форматом обмена RFC 7049. Документ содержит редакционные правки, дополнительные детали и исправления ошибок. Документ не задаёт новую версию формата.

G.1. Ошибки и редакционные правки

Исправлены две подтверждённые ошибки в RFC 7049 [Err3764] [Err3770] (параграф 3.4.3: "29" -> "49", параграф 5.5: "0b000_11101" -> "0b000_11001"). Кроме того, в RFC 7049 дан пример использования числа 24 для простого значения [Err5917], что является ошибкой формы - пример исключён. В [Err5763] указана ошибка в формулировке определения тега, она исправлена в формулировке параграфа 3.4. В [Err5434] отмечено, что пример Universal Binary JSON (UBJSON) в Приложении E больше не соответствует текущей версии UBJSON на момент обнаружения ошибки. Оказалось, что спецификация UBJSON была полностью изменена с 2013 г., поэтому пример был удалён. В [Err4409] [Err4963] [Err4964] отмечена обременительность правил сортировки ключей отображения для канонического кодирования - это привело к пересмотру предложений по каноническому кодированию и замене их предложениями по детерминированному кодированию (см. ниже). Учтено сообщение [Err4294] путём добавления второго значения в комментарий к последнему примеру в параграфе 3.2.2, улучшившего симметрию.

Другие редакционные правки указаны ниже.

- Используется новая функциональность xml2rfc [RFC7991].
- Дополнительно разъяснена используемая нотация.
- Обновлены ссылки, например, [RFC8259] вместо RFC 4627, [RFC7228] вместо CNN-TERMS и 11-е издание [ECMA262] вместо 5.1. Добавлены ссылки на [IEEE754] и включены необходимые определения, добавлены ссылки на [C] и [Cplusplus20], а также ссылка на [RFC8618], где дополнительно иллюстрируется обсуждение из Приложения E.
- При обсуждении диагностической нотации (раздел 8) упомянута расширенная диагностическая нотация (Extended Diagnostic Notation или EDN), определённая в [RFC8610], выделен пробел в представлении данных NaN, а также добавлено разъяснения представления строк неопределённого размера без блоков (параграф 8.1).
- Добавлено это приложение.

G.2. Изменения взаимодействия с IANA

Обновлено описание взаимодействия с IANA (редакционные правки, например указание рабочей группы CBOR как автора спецификации). Добавлены ссылки на реестры IANA в раздел 11.2. Дополнительная литература.

В реестре Concise Binary Object Representation (CBOR) Tags [IANA.cbor-tags] теги 256 - 32767 (нижняя половина 1+2) больше не назначаются по процедуре First Come First Served, заменённой на процедуру Specification Required.

G.3. Изменения в предложениях и других информационных компонентах

При пересмотре документа, помимо исправления замеченных ошибок, рабочая группа учла почти 7-летний опыт применения CBOR в разнообразных приложениях. Это привело к ряду редакционных изменений, включая добавление таблиц для иллюстрации, а также выделение одних аспектов и снижение внимания к другим.

Важным дополнением является раздел 2 с обсуждением модели данных CBOR и её небольших вариаций, связанных с обработкой CBOR. Введение терминов для этих вариаций (базовая, расширенная базовая, конкретная) позволило сократить текст в других частях документа и помогло прояснить ожидания от реализаций и возможностей расширения.

Поскольку формат был выведен из экосистемы JSON, на RFC 7049 существенно повлияла система чисел в JSON, которая в свою очередь была унаследована от JavaScript. В JSON не разделяются целочисленные (integer) и действительные (floating-point) значения (последние указываются в десятичной форме). В CBOR применяется двоичное представление чисел, различающее значения integer и floating-point. Опыт реализации и развёртывания показал, что такое разделение следует более чётко указать в документе - текст, предполагающий простую замены действительного числа целым, был исключён. Кроме того, добавлено предложение (на основе I-JSON [RFC7493]) по обработке этих типов при преобразовании JSON в CBOR, а также рекомендовано использовать конкретный механизм округления.

В модели данных CBOR часто предполагает разные варианты кодирования одного значения. Новый раздел 4 вводит термин «предпочтительная сериализация» (preferred serialization, параграф 4.1) и определяет его для разных типов элементов данных. На основе этого в разделе обсуждается, как протокол на основе CBOR может определить детерминированное кодирование (параграф 4.2), чтобы избежать терминов «канонический» (canonical) и «канонизация» из RFC 7049. Параграф 4.2.1. Базовые требования детерминированного кодирования разрешает базовую поддержку таких протокольных требований к кодированию. Этот документ дополнительно упрощает реализацию детерминированного кодирования за счёт замены порядка отображений, предложенного в RFC 7049, простым лексикографическим порядком кодированных ключей. Описание старого предложения сохранено как вариант, названный упорядочением сначала по размеру (параграф 4.2.3).

Уточнена и более строго применяется терминология для корректно сформированных и действительных (пригодных) данных, что позволило избежать в примерах менее чётких терминов «синтаксическая ошибка» (syntax error), «ошибка декодирования» (decoding error» и «строгий режим» (strict mode). Явно назван третий уровень требования приложения к вводу, выходящий за пределы проверки пригодности на уровне CBOR. Корректно сформированный (пригодный для обработки), действительный (проверенный базовым декодером на пригодность) и ожидаемый (проверенный приложением) ввод образуют иерархию уровней пригодности (применимости).

Обработка простых значений с некорректной формой разъяснена в тексте и псевдокоде. Добавлено Приложение F с обсуждением ошибок формы и примерами. Псевдокод обновлён для улучшения переносимости и рассмотрены вопросы переносимости.

Обсуждение пригодности поделено на 2 части. Разъяснена пригодность отображений (обработка дубликатов ключей) и сфера применимости некоторых реализаций. При обсуждении терминологии для тегов, их номеров и содержимого добавлена пригодность тегов и разъяснены ограничения для содержимого в целом и конкретно для тега 1.

В параграф 3.4 добавлено примечание для разработчиков (и определения новых тегов) об определении тегов с зависящей от порядка сериализации семантикой.

Тег 35 не определён в этом документе, регистрация на основе определения из RFC 7049 сохранена.

В разделе 3 определены термины «аргумент» (argument) и «голова» (head), упрощающие дальнейшее обсуждение.

Раздел 10. Вопросы безопасности по большей части переписан, во многих местах документа теперь явно указано, что декодер не может просто мириться с ошибками формы.

Благодарности

Прообразом CBOR является формат MessagePack, разработанный и продвигаемый Sadayuki Furuhashi ("frsyuki"). Это упоминание MessagePack предназначено лишь для указания авторства, CBOR не является версией или заменой MessagePack, поскольку они различаются по целям и требованиям.

Потребность в функциях, выходящих за пределы исходной спецификации MessagePack, стала очевидной для многих примерно в 2012 г. Формат BinaryPack на основе небольшого изменения MessagePack разработал Eric Zhang для проекта binaryjs. Похожее, но своё расширение предложил Tim Caswell для проектов msgpack-js и msgpack-js-browser. Многие люди участвовали в обсуждении расширения MessagePack для разделения представлений строк текста и байтов.

Прообразом кодирования дополнительной информации в CBOR послужило кодирование сведений о размере, разработанное Klaus Hartke для CoAP.

Этот документ включает предложения многих людей, среди которых выделяются Dan Frost, James Manger, Jeffrey Yasskin, Joe Hildebrand, Keith Moore, Laurence Lundblade, Matthew Lepinski, Michael Richardson, Nico Williams, Peter Occil, Phillip Hallam-Baker, Ray Polk, Stuart Cheshire, Tim Bray, Tony Finch, Tony Hansen и Yaron Sheffer. Benjamin Kaduk представил обширный обзор в процессе обработки IESG. Éric Vyncke, Erik Kline, Robert Wilton и Roman Danyliw предоставили дополнительные комментарии IESG, в том числе обзор директората IoT от Eve Schooler.

Адреса авторов

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Paul Hoffman
ICANN
Email: paul.hoffman@icann.org

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru