Internet Engineering Task Force (IETF)
Request for Comments: 9651
Obsoletes: 8941
Category: Standards Track
ISSN: 2070-1721

M. Nottingham
Cloudflare
P-H. Kamp
The Varnish Cache Project

September 2024

Structured Field Values for HTTP

Значения структурированных полей для HTTP

Аннотация

В этом документе описывается набор данных и связанных с ними алгоритмов, предназначенных для упрощения и повышения безопасности при задании и обработке полей заголовков и трейлеров HTTP, известных как структурированные (Structured Fields, Structured Headers, Structured Trailers). Документ предназначен для использования в спецификациях новых полей HTTP.

Документ заменяет собой RFC 8941.

Статус документа

Документ содержит проект стандарта 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.

Информацию о текущем статусе документа, ошибках и способах обратной связи можно найти по ссылке https://www.rfc-editor.org/info/rfc9651.

Оглавление

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

1. Введение

Задание синтаксиса новых полей в заголовках и трейлерах HTTP является трудоёмкой задачей даже с учётом рекомендаций параграфа 16.3.2 в [HTTP], поскольку авторам придётся принять множество решений и столкнуться с подводными камнями.

Как только поле определено, часто возникает необходимость создания новых синтаксических анализаторов и средств сериализации (преобразования), поскольку обработка каждого значения несколько отличается от базового синтаксиса.

В этом документе вводится базовый набор структур данных для использования в определениях новых полей HTTP с учётом отмеченных проблем. В частности, определяется базовая абстрактная модель, а также конкретная сериализация для выражения этой модели в полях заголовков и трейлеров HTTP [HTTP].

Поле HTTP, заданное как Structured Header или Structured Trailer (или Structured Field, если оно может включаться и в заголовки и в трейлеры), использует заданные здесь типы для определения своего синтаксиса и базовых правил обработки, что упрощает как определение полей разработчиками спецификаций, так и их обработку в реализациях. Будущие версии HTTP могут задавать дополнительные варианты сериализации абстрактной модели этих структур, позволяя передавать использующие модель поля более эффективно без необходимости переопределять их.

Отметим, что переопределение синтаксиса имеющихся полей HTTP не является целью этого документа и описанные здесь механизмы предназначены лишь для полей, явно выбравших эти механизмы.

В разделе 2 указано, как задать структурированное поле, а в разделе 3 определены абстрактные типы данных для использования в структурированных полях. Эти абстрактные типы могут быть преобразованы в поля HTTP или извлечены из таких полей с использованием алгоритмов, описанных в разделе 4.

1.1. Преднамеренно строгая обработка

В спецификации намеренно задан строгий синтаксический анализ и сериализация с использованием пошаговых алгоритмов и единственным вариантом обработки ошибок является отказ всей операции целиком. Это предназначено для обеспечения точности реализации и функциональной совместимости. Реализация, пытающаяся быть полезной за счёт повышения толерантности к входным данным, ухудшит функциональную совместимость, поскольку будет создавать давление на другие реализации с целью поддержки аналогичных (вероятно, слегка отличающихся) путей обхода. Иными словами, строгость обработки является преднамеренным свойством данной спецификации, позволяющим обнаруживать и исправлять ошибки на входе для предотвращения проблем совместимости и безопасности, которые могли бы возникнуть.

Отметим, что в результате такой строгости ошибка одной стороны при вводе данных несколькими сторонами (например, посредниками или другими компонентами отправителя) скорей всего вызовет отказ при синтаксическом анализе значения поля.

1.2. Соглашения о нотации

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

В документе применяются правила VCHAR, SP, DIGIT, ALPHA и DQUOTE из [RFC5234] для задания чисел и соответствующих байтов ASCII, в зависимости от контекста. Для этого же служат правила tchar и OWS из [HTTP].

В документе используются алгоритмы, задающие поведение при синтаксическом анализе и сериализации. При анализе полей HTTP реализации **должны** обеспечивать поведение, не отличимое от заданного алгоритмами.

Для сериализации полей HTTP алгоритмы задают рекомендуемый способ выполнения. Реализации **могут** отклоняться от заданного поведения при условии, что результаты по-прежнему корректно обрабатываются алгоритмами синтаксического анализа, описанными в параграфе 4.2.

2. Определение новых структурированных полей

Для задания поля HTTP как структурированного (Structured Field) авторы должны:

- указать нормативную ссылку на этот документ, которая будет вносить указанные здесь требования для создателей и получателей поля;
- указать категорию поля Structured Header (может включаться только в заголовки - базовый вариант), Structured Trailer (только трейлеры) или Structured Field (оба варианта);
- указать тип значения поля - List (параграф 3.1), Dictionary (параграф 3.2) или Item (параграф 3.3);
- определить семантику значения поля;
- задать дополнительные ограничения для значения поля, а также последствия их нарушения.

Обычно это будет означать указание в определении поля типа верхнего уровня (List, Dictionary, Item), а также определение допустимых типов и ограничения для них. Например, поле типа List может содержать лишь элементы типа Integer или сочетание типов, заголовок, заданный как Item, позволяет включать элементы String, причём лишь строки, начинающиеся с буквы Q, или строки в нижнем регистре. Аналогично, Inner List (параграф 3.1.1) действительны лишь в том случае, когда это явно разрешено определением поля. Для полей Display String рекомендуется чётко указывать разрешённые коды Unicode, например, использование профиля из [PRECIS].

В определениях полей эта спецификация может применяться для поля целиком, а не для отдельных его частей. Спецификации могут указывать имя поля как Structured Header name, Structured Trailer name или Structured Field name. Значения могут указываться как Structured Header value, Structured Trailer value или Structured Field value.

Эта спецификация задаёт минимальные размеры или число различных структур, поддерживаемых реализациями. Максимальные размеры обычно не задаются, но авторам следует понимать, что разные реализации HTTP могут вносить свои ограничения для размеров отдельных полей и/или полного размера заголовков или трейлеров.

2.1. Пример

Вымышленное поле заголовка Foo-Example можно задать, как показано ниже.

42. Поле заголовка Foo-Example

    Поле Foo-Example в заголовке HTTP переносит сведения о количестве Foo в сообщении.

    Foo-Example - это структурированное поле заголовка [RFC9651]. Его значением **должно** быть целое число (Integer, параграф 3.3.1 в [RFC9651]).

    Значение поля указывает количество Foo в сообщении и **должно** относиться к диапазону от 0 до 10, включительно. При иных значениях поле заголовка **должно** игнорироваться целиком.

    Определён указанный ниже параметр.

    - Параметр с ключом foourl и значением типа String (параграф 3.3.3 в [RFC9651]) передаёт Foo URL для сообщения. Требования к обработке указаны ниже.

      Параметр foourl содержит ссылку URI (параграф 4.1 в [RFC3986]). Если значение параметра не является действительной ссылкой URI, поле заголовка **должно** игнорироваться целиком. Значения, являющиеся относительными ссылками (параграф 4.2 в [RFC3986]), **должны** преобразовываться (resolve, раздел 5 в [RFC3986]) до использования.

    Пример поля показан ниже.

```
Foo-Example: 2; foourl="https://foo.example.com/"
```

2.2. Обработка ошибок

При сбое синтаксического анализа игнорируется поле целиком (параграф 4.2). Определения полей не могут переопределять это правило, поскольку это будет препятствовать обработке программами общего назначения. Разрешается лишь вводить дополнительные ограничения (например, численные значения Integer и Decimal, формат String и Token, типы, разрешённые в значениях Dictionary, число Item в List).

При нарушении ограничений, относящихся к конкретному полю, такое поле также игнорируется целиком, если определение не задаёт иное. Например, если поле заголовка задано как Item и должно иметь тип Integer, но получено значение String, поле следует игнорировать, если в определении поля явно не задано иное поведение.

2.3. Сохранение расширяемости

Структурированные поля спроектированы так, чтобы их можно было расширять, поскольку опыт показывает, что часто возникает необходимость изменения и дополнения допустимого синтаксиса, даже если это не предполагалось.

Типы Item и Inner List позволяют в качестве средства расширения использовать параметры. Это означает, что при необходимости значения могут быть расширены для добавления новой информации. Для совместимости с новыми версиями в спецификациях полей не рекомендуется считать ошибкой присутствие нераспознанного параметра.

Для сохранения расширяемости поля должны иметь тип Item, List или Dictionary. Поля, ошибочно заданные с иным типом (например, Integer), считаются полями типа Item (т. е., поддерживающими параметры).

Для гарантии расширяемости в будущем и стимулирования использования полных реализаций синтаксических анализаторов определение поля может указывать возможность добавления параметров отправителем. В спецификациях можно предусмотреть резервирование всех параметров, соответствующих определённому шаблону, которые затем могут включаться в ту или иную часть запроса. Это будет препятствовать созданию синтаксических анализаторов, не учитывающих параметры.

Спецификации, использующие Dictionary, могут обеспечивать совместимость с будущими версиями, требуя игнорировать неизвестные ключи, а также связанные с ними значения и типы. Это позволит будущим спецификациям добавлять ключи, устанавливая для них соответствующие разрешения.

Расширение структурированного поля может требовать от получателя полностью игнорировать понятное тому расширение, если в нем не соблюдаются заданные для значения ограничения.

2.4. Использование новых структурированных типов в расширениях

Поскольку определение поля должно ссылаться на конкретный RFC для Structured Field, типы, доступные в качестве значения поля, ограничены указанными в данном RFC. Например, поле, определение которого ссылается на этот документ, может использовать значение с типом Date (параграф 3.3.7), а поле, ссылающееся на RFC 8941, не может, поскольку оно будет считаться недействительным (и отбрасываться) реализациями этой спецификации. Это ограничение применимо и к будущим расширениям поля, например, поле, заданное со ссылкой на RFC 8941, не может использовать тип Date, поскольку некоторые получатели могут применять для его обработки анализатор, основанный на RFC 8941. Однако этот документ разработан с учётом совместимости с RFC 8941 и анализатор, реализующий представленные здесь требования, сможет разбирать действительные структурированные поля, определения которых ссылаются на RFC 8941.

Обновление реализации структурированных полей для поддержки нового выпуска спецификации (такого как этот документ) ведёт к тому, что некоторые значения полей, недействительные в соответствии с прежним RFC, могут стать действительными. Например, экземпляр поля может содержать синтаксически корректное значение Date (параграф 3.3.7), даже если определение этого поля не включает Date. Реализация на основе RFC 8941 будет отвергать такие экземпляры, поскольку они не заданы в спецификации, но если её обновить в соответствии с данной спецификацией, разбор поля будет успешным. В некоторых случаях результирующее значение Date будет отклонено логикой поля, но значения, которые в ином случае игнорировались бы (например, как значения параметров), могут быть не обнаружены и поле впоследствии может быть воспринято и обработано.

3. Структурированные типы данных

В этом разделе представлен обзор абстрактных типов, используемых в структурированных полях, а также краткое описание и примеры преобразования этих типов в текстовые поля HTTP. В разделе 4 подробно описан синтаксический анализ и преобразование в текстовые поля HTTP.

- Имеется три типа верхнего уровня, которые могут быть определены как поля HTTP: List, Dictionary и Item.
- List и Dictionary - это контейнеры, элементами которых могут быть Item или Inner List (массив Item).
- Item и Inner List могут быть параметризованы парами ключ-значение.

3.1. List

Список - это массив (возможно, пустой) элементов, которыми могут быть Item (параграф 3.3) или Inner List (параграф 3.1.1), те и другие могут быть параметризованными (параграф 3.1.2). Пустой список указывается тем, что поле не преобразуется. Это означает, что поля, заданные как List по умолчанию имеют пустое значение. При преобразовании списка в текстовое поле HTTP элементы разделяются запятыми и (необязательными) пробелами. Например, поле, заданное как список (List) маркеров (Token), может иметь вид

```
Example-List: sugar, tea, rum
```

Отметим, что элементы List можно разбивать на несколько строк одного заголовка или трейлера, как указано в параграфе 5.3 [HTTP]. Например, строка

```
Example-List: sugar, tea, rum
```

эквивалентна строкам

```
Example-List: sugar, tea
Example-List: rum
```

Однако отдельные элементы списка нельзя безопасно разбить на несколько строк (см. параграф 4.2).

Синтаксические анализаторы **должны** поддерживать списки по меньшей мере до 1024 элементов. Спецификации полей могут ограничивать типы и число элементов в отдельных значениях List при необходимости.

3.1.1. Inner List

Inner List может содержать элементы Item (параграф 3.3) или быть пустым. Как отдельные Item, так и Inner List могут быть параметризованы (параграф 3.1.2).

При преобразовании в текстовое поле HTTP списки Inner List заключаются в круглые скобки, а их значения разделяются одним или несколькими пробелами. Поле, значение которого определено как список (List) из Inner List, состоящих из строк (String), может иметь вид

```
Example-List: ("foo" "bar"), ("baz"), ("bat" "one"), ()
```

Отметим, что последним элементом в этом примере является пустой Inner List.

Поле заголовка, значение которого задано как список Inner List с параметрами (Parameters) на обоих уровнях, может иметь вид

```
Example-List: ("foo"; a=1;b=2);lvl=5, ("bar" "baz");lvl=1
```

Синтаксические анализаторы **должны** поддерживать Inner List с 256 элементами по меньшей мере. Спецификации полей могут ограничивать типы и число отдельных элементов Inner List.

3.1.2. Параметры

Параметры - это упорядоченный набор пар ключ-значение, связанных с Item (параграф 3.3) или Inner List (параграф 3.1.1). Ключи в наборе уникальны, а значения являются простыми элементами (т. е. не могут быть параметризованы, см. параграф 3.3).

Реализации **должны** обеспечивать доступ к параметрам как по индексу, так и по ключу. В спецификациях **можно** использовать любой из этих методов доступа.

Отметим, что параметры являются упорядоченными, а ключи не могут содержать букв верхнего регистра.

При преобразовании в текстовое поле HTTP параметр отделяется от Item или Inner List и других параметров точкой с запятой (;), например,

```
Example-List: abc;a=1;b=2; cde_456, (ghi;jk=4 l);q="9";r=w
```

В параметрах с логическим (Boolean, параграф 3.3.6) значением true это значение **должно** исключаться при преобразовании (сериализации). Например, при a = true и b = false запись будет иметь вид

```
Example-Integer: 1; a; b=?0
```

Это требование относится лишь к преобразованию и анализаторы **должны** корректно обрабатывать значение true.

Синтаксические анализаторы **должны** поддерживать по меньшей мере до 256 параметров на Item или Inner List, а также поддерживать ключи параметров размером по меньшей мере до 64 символов. Спецификации полей могут ограничивать порядок отдельных параметров, а также типы их значений.

3.2. Dictionary

Словарь представляет собой упорядоченный набор пар ключ-значение, где ключ является короткой строкой текста, а значение имеет тип Item (параграф 3.3) или содержит массив Item (в обоих случаях могут применяться параметры, параграф 3.1.2). Словарь может быть пустым, а ключи уникальны в рамках конкретного словаря.

Реализации **должны** обеспечивать доступ к словарю как по индексу, так и по ключу. В спецификациях **можно** использовать любой из этих методов доступа.

Как и списки, пустой словарь представляет исключением всего поля. Это означает, что поля типа Dictionary по умолчанию имеют пустое значение.

Обычно спецификация поля задаёт семантику словарей, указывая разрешённые типы элементов по их ключам, а также обязательность или необязательность присутствия. Получатели **должны** игнорировать элементы с неопределёнными или неизвестными ключами, если спецификация поля специально на запрещает это.

При преобразовании в текстовое поле HTTP элементы упорядочиваются по результатам преобразования и разделяются запятыми с необязательным пробелом. Ключи не могут содержать символов верхнего регистра, между ключом и значением указывается знак равенства (=, без пробела). Например,

```
Example-Dict: en="Applepie", da=:w4ZibGV0w6ZydGU=:
```

В этом примере символ = в конце относится к последовательности байтов (Byte Sequence, параграф 3.3.5).

В параметрах с логическим (параграф 3.3.6) значением true это значение **должно** исключаться при преобразовании. Например, при b и c со значением true запись будет иметь вид

```
Example-Dict: a=?0, b, c; foo=bar
```

Это требование относится лишь к преобразованию и анализаторы **должны** корректно обрабатывать значение true.

Пример словаря с элементами типа Inner List из Token

```
Example-Dict: rating=1.5, feelings=(joy sadness)
```

Словарь с сочетанием Item и Inner List, часть которых имеет параметры может иметь вид

```
Example-Dict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid
```

Словари могут разбивать свои элементы на несколько строк одного заголовка или трейлера. Например,

```
Example-Dict: foo=1, bar=2
```

эквивалентно

```
Example-Dict: foo=1
Example-Dict: bar=2
```

Однако отдельные элементы Dictionary нельзя разбивать по строкам (см. параграф 4.2).

Синтаксические анализаторы **должны** поддерживать словари с 1024 парами ключ-значение по меньшей мере. Спецификации полей могут порядок и типы значений отдельных элементов Dictionary.

3.3. Item

Item может быть целым числом (Integer, параграф 3.3.1), десятичным значением (Decimal, параграф 3.3.2), строкой (String, параграф 3.3.3), маркером (Token, параграф 3.3.4), последовательностью байтов (Byte Sequence, параграф 3.3.5), логическим значением (Boolean, параграф 3.3.6), датой (Date, параграф 3.3.7) или Display String (параграф 3.3.8) и включать параметры (параграф 3.1.2). Например, поле заголовка, заданное как Item типа Integer может иметь вид

```
Example-Integer: 5
```

или включать параметры

```
Example-Integer: 5; foo=bar
```

3.3.1. Integer

Целые числа имеют значения от -999999999999999 до 999999999999999, включительно (до 15 цифр и знак), для совместимости с IEEE 754 [IEEE754]. Например,

```
Example-Integer: 42
```

Числа с количеством знаков больше 15 могут поддерживаться разными способами, например, с использованием String (параграф 3.3.3), Byte Sequence (параграф 3.3.5) или параметра типа Integer, указывающего коэффициент умножения.

Хотя допускается сериализация чисел с нулями в начале (например, 0002, -01) и нуля со знаком (-0), эти тонкости могут не поддерживаться реализациями.

3.3.2. Decimal

Decimal - это число с целой и дробной частью. Целая часть может иметь до 12 цифр, дробная - не более 3. Например, заголовок с полем Decimal может иметь вид

```
Example-Decimal: 4.5
```

Хотя допускается сериализация Decimal с нулями в начале (например, 0002.5, -01.334), в конце (например, 5.230, -0.40) и нулей со знаком (например, -0.0), эти тонкости могут не поддерживаться реализациями.

Отметим, что алгоритмы преобразования (параграф 4.1.5) округляют входные значения, сохраняя в дробной части не более 3 цифр. Если нужен иной вариант округления, его следует указать до выполнения сериализации.

3.3.3. String

Строка содержит печатаемые символы ASCII [RFC0020] (%x20 - %x7E) и может быть пустой. Отметим, что набор символов не включает табуляцию, перевод строки, возврат каретки и т. п. Символы других кодировок напрямую не поддерживаются в String, поскольку вызывают проблемы совместимости и (за редкими исключениями) не требуются. Если же такие символы нужны, можно использовать Display String (параграф 3.3.8).

При преобразовании в текстовое поле HTTP строки заключаются в двойные кавычки с использованием символа обратной дробной черты (\) для экранирования кавычек и символов \. Например,

```
Example-String: "hello world"
```

Отметим, что в строках применяется только разграничитель DQUOTE, а одинарные кавычки не указывают границу String. Кроме того, символы DQUOTE и \ могут экранироваться, а любой другой символ после \ **должен** вызывать отказ при синтаксическом анализе. Анализаторы **должны** поддерживать String размером (после декодирования) по меньшей мере 1024 символа.

3.3.4. Token

Маркерами (Token) называют короткие текстовые слова, начинающиеся с буквы или символа *, за которым могут следовать символы-маркеры, разрешённые правилом ABNF token из [HTTP], плюс символы : и /. Например,

```
Example-Token: foo123/456
```

Анализаторы **должны** поддерживать Token размером по меньшей мере 512 символов.

Отметим, что маркеры определены в основном для совместимости с моделью данных имеющихся полей HTTP и для их использования в некоторых реализациях могут потребоваться дополнительные действия. Поэтому в новых полях рекомендуется использовать String.

3.3.5. Byte Sequence

В структурированных полях можно передавать последовательности байтов (Byte Sequence).

При преобразовании в текстовое поле HTTP последовательности байтов ограничиваются двоеточиями (:) и кодируются в формате base64 (раздел 4 в [RFC4648]). Например,

```
Example-ByteSequence: :cHJldGVuZCB0aGlzIGlzIGJpbmFyeSBjb250ZW50Lg==:
```

Анализаторы **должны** поддерживать Byte Sequence размером (после декодирования) по меньшей мере 16384 октета.

3.3.6. Boolean

В структурированных полях могут передаваться логические значения (Boolean).

При преобразовании в текстовое поле HTTP значения Boolean указываются префиксом ?, за которым следует 1 для true или 0 для false. Например,

```
Example-Boolean: ?1
```

Отметим, что в Dictionary (параграф 3.2) и Parameter (параграф 3.1.2) логическое значение true исключается.

3.3.7. Date

В структурированных полях могут передаваться значения дат (Date), для которых применяется модель данных, аналогичная Integer, с представлением числа (возможно отрицательного) секунд после полуночи 1 января 1970 г. (1970-

01-01T00:00:00Z) без учёта високосных секунд. Преобразование Date в текстовые поля HTTP аналогично применяемому для Integer, но используется префикс @. Например,

```
Example-Date: @1659578233
```

Анализаторы **должны** поддерживать Date, значения которых включают все дни в годах с 1 до 9999 (т. е. от -62135596800 до 253402214400 секунд от полуночи 1 января 1970 г.).

3.3.8. Display String

Тип Display String похож на String, но позволяет использовать скалярные значения Unicode (т. е. все коды Unicode кроме суррогатных). Тип Display String предназначен для случаев, когда значение представляется пользователю и может содержать символы, отличные от ASCII. **Не рекомендуется** применять этот тип в ситуациях, где подойдёт String (параграф 3.3.3) или Token (параграф 3.3.4), поскольку для Unicode имеются требования по обработке (например, нормализация) и безопасности (например, атаки с омографами), способные осложнить обработку. Отметим, что в Display String не указывается применяемый язык, при необходимости это можно сделать отдельно (например, с использованием параметра).

В текстовых поля HTTP представление Display String похоже на String, но используется префикс % и для символов, не относящихся к ASCII, применяется %-кодирование, например,

```
Example-DisplayString: %"This is intended for display to %c3%bcsers."
```

Вопросы безопасности при обработке Display String рассматриваются в разделе 6.

4. Работа со структурированными полями в HTTP

В этом разделе определено преобразование (сериализация) и синтаксический анализ (разбор) абстрактных типов, заданных в разделе 3, в значения текстовых полей HTTP и другие совместимые кодировки (например, в HTTP/2 [HTTP/2] до сжатия с помощью HPACK [HPACK]).

4.1. Преобразование структурированных полей

Для заданной спецификацией структуры возвращается строка ASCII, подходящая в качестве значения поля HTTP.

1. Если структура имеет тип Dictionary или List с пустым значением (нет элементов), поле не преобразуется совсем, т. е. исключаются field-name и field-value.
2. Если структура имеет тип List, output_string будет результатом преобразования List (параграф 4.1.1).
3. Если структура имеет тип Dictionary, output_string будет результатом преобразования Dictionary (параграф 4.1.2).
4. Если структура имеет тип Item, output_string будет результатом преобразования Item (параграф 4.1.3).
5. Иначе сериализация завершается отказом.
6. Возвращается значение output_string в форме строки байтов с кодировкой ASCII [RFC0020].

4.1.1. List

Для массива кортежей (member_value, parameters) как input_list возвращается строка ASCII, пригодная для поля HTTP.

1. В качестве выходного значения принимается пустая строка.
2. Для каждого кортежа (member_value, parameters) из input_list выполняются следующие операции.
    1. Если member_value является массивом, в конец выходной строки добавляется результат преобразования Inner List (параграф 4.1.1.1) с (member_value, parameters).
    2. Иначе в конец выходной строки добавляется результат преобразования Item (member_value, parameters) (параграф 4.1.3).
    3. Если в input_list ещё есть member_value:
        1. в конец выходной строки добавляется запятая (,);
        2. в конец выходной строки добавляется пробел (SP).
3. Возвращается выходная строка.

4.1.1.1. Inner List

Для массива кортежей (member_value, parameters) как inner_list и списка параметров в качестве list_parameters возвращается строка ASCII, пригодная для поля HTTP.

1. В качестве выходного значения принимается открывающая круглая скобка «(«.
2. Для каждого кортежа (member_value, parameters) из inner_list выполняются следующие операции.
    1. В конец выходной строки добавляется результат преобразования Item (member_value, parameters) (параграф 4.1.3).
    2. Если в inner_list ещё остаются значения, в конец выходной строки добавляется пробел (SP).
3. В конец выходной строки добавляется закрывающая круглая скобка «)».
4. В конец выходной строки добавляется результат обработки параметров list_parameters (параграф 4.1.1.2).
5. Возвращается выходная строка.

4.1.1.2. Параметры

Для упорядоченного словаря input_parameters (каждый элемент имеет param_key и param_value) возвращается строка ASCII, пригодная для поля HTTP.

1. В качестве выходного значения принимается пустая строка.
2. Для каждого param_key со значением param_value из input_parameters выполняются следующие операции.
    1. В конец выходной строки добавляется точка с запятой (;).
    2. В конец выходной строки добавляется результат преобразования ключа param_key (параграф 4.1.1.3).
    3. Если param_value не является логическим (Boolean) значением true:
        1. В конец выходной строки добавляется знак равенства (=);
        2. В конец выходной строки добавляется результат преобразования param_value (параграф 4.1.3.1).
3. Возвращается выходная строка.

4.1.1.3. Ключи

Для ключа input_key возвращается строка ASCII, пригодная для поля HTTP.

1. Значение input_key преобразуется в последовательность символов ASCII. При отказе преобразование прерывается.
2. Если input_key содержит символы, отличные от lcalpha[1], DIGIT, «_», «-», «.», «*», преобразование прерывается.
3. Если первый символ input_key отличается от lcalpha и «*», преобразование прерывается.
4. В качестве выходного значения принимается пустая строка.
5. В конец выходной строки добавляется input_key (после преобразований).
6. Возвращается выходная строка.

4.1.2. Dictionary

Для упорядоченного словаря input_dictionary (каждый элемент имеет member_key и кортеж (member_value, parameters)) возвращается строка ASCII, пригодная для поля HTTP.

1. В качестве выходного значения принимается пустая строка.
2. Для каждого member_key со значением (member_value, parameters) из input_dictionary выполняются действия:
    1. В конец выходной строки добавляется результат преобразования ключа member_key (параграф 4.1.1.3).
    2. Если member_value является логическим (Boolean) значением true:
        1. В конец выходной строки добавляется результат преобразования параметров (параграф 4.1.1.2).
    3. Иначе:
        1. В конец выходной строки добавляется знак равенства (=);
        2. Если member_value является массивом, в конец выходной строки добавляется результат преобразования Inner List (member_value, parameters) (параграф 4.1.1.1);
        3. Иначе в в конец выходной строки добавляется результат преобразования Item (member_value, parameters) (параграф 4.1.3).
    4. Если в input_dictionary ещё имеются элементы:
        1. В конец выходной строки добавляется запятая (,).
        2. В конец выходной строки добавляется пробел (SP).
3. Возвращается выходная строка.

4.1.3. Item

Для элемента bare_item и параметров item_parameters возвращается строка ASCII, пригодная для поля HTTP.

1. В качестве выходного значения принимается пустая строка.
2. В конец выходной строки добавляется результат преобразования простого элемента bare_item (параграф 4.1.3.1).
3. В конец выходной строки добавляется результат преобразования параметров item_parameters (параграф 4.1.1.2).
4. Возвращается выходная строка.

4.1.3.1. Простой элемент (Bare Item)

Для элемента input_item возвращается строка ASCII, пригодная для поля HTTP.

1. Если input_item является Integer, возвращается результат преобразования Integer (параграф 4.1.4) для input_item.

[1]Латинские буквы нижнего регистра (a - z). *Прим. перев.*

2. Если input_item является Decimal, возвращается результат преобразования Decimal (параграф 4.1.5) для input_item.
3. Если input_item является String, возвращается результат преобразования строки input_item (параграф 4.1.6).
4. Если input_item является Token, возвращается результат преобразования маркера input_item (параграф 4.1.7).
5. Если input_item является Byte Sequence, возвращается результат преобразования Byte Sequence input_item (параграф 4.1.8).
6. Если input_item является Boolean, возвращается результат преобразования Boolean для input_item (параграф 4.1.9).
7. Если input_item является Date, возвращается результат преобразования даты input_item (параграф 4.1.10).
8. Если input_item является Display String, r возвращается результат преобразования Display String input_item (параграф 4.1.11).
9. В противном случае преобразование завершается отказом.

4.1.4. Integer

Для целого числа input_integer возвращается строка ASCII, пригодная для поля HTTP.

1. Если input_integer выходит за пределы диапазона -999999999999999 - 999999999999999 (включительно), преобразование завершается отказом.
2. В качестве выходного значения принимается пустая строка.
3. Если input_integer меньше 0 (отрицательно), в конец выходного значения добавляется знак минус (-).
4. В конец выходного значения добавляется десятичное представление (только цифры).
5. Возвращается выходная строка.

4.1.5. Decimal

Для десятичного числа input_decimal возвращается строка ASCII, пригодная для поля HTTP.

1. Если input_decimal не является десятичным числом, преобразование завершается отказом.
2. Если input_decimal имеет более 3 значащих цифр после десятичной точки, дробная часть округляется до 3 знаков с использованием в последнем знаке ближайшей или чётной цифры (при равноудалённости).
3. Если input_decimal имеет более 12 значащих цифр слева от десятичной точки, преобразование завершается отказом.
4. В качестве выходного значения принимается пустая строка.
5. Если input_decimal меньше 0 (отрицательно), в конец выходного значения добавляется знак минус (-).
6. В конец выходного значения добавляется десятичное представление целой части input_decimal (только цифры) или 0, если целая часть равна 0 (отсутствует).
7. В конец выходного значения добавляется точка (.).
8. Если дробная часть равна 0, в конец выходного значения добавляется 0.
9. Иначе в конец выходного значения добавляется десятичное представление дробной части после округления (только цифры).
10. Возвращается выходная строка.

4.1.6. String

Для строки input_string возвращается строка ASCII, пригодная для поля HTTP.

1. Значение input_string представляется строкой символов ASCII, при отказе всё преобразование завершается отказом.
2. Если input_string содержит символы из диапазона %x00-1f или %x7f-ff (т. е., не VCHAR или SP), преобразование завершается отказом.
3. В качестве выходного значения принимается DQUOTE (").
4. Для каждого символа char из input_string выполняются следующие операции:
    1. Если char имеет значение \ или DQUOTE:
        1. В конец выходного значения добавляется символ \.
    2. В конец выходного значения добавляется char.
5. В конец выходного значения добавляется DQUOTE.
6. Возвращается выходная строка.

4.1.7. Token

Для маркера input_token возвращается строка ASCII, пригодная для поля HTTP.

1. Строка input_token преобразуется в последовательность символов ASCII, при отказе всё преобразование завершается отказом.
2. Если первый символ input_token не относится к ALPHA или «*» или остальная часть включает символ, не относящийся к tchar, «:», «/», преобразование завершается отказом.
3. В качестве выходного значения принимается пустая строка.
4. В конец выходного значения добавляется input_token.
5. Возвращается выходная строка.

4.1.8. Byte Sequence

Для последовательности байтов input_bytes возвращается строка ASCII пригодная для использования в поле HTTP.

1. Если input_bytes не является последовательностью байтов, преобразование завершается отказом.
2. В качестве выходного значения принимается пустая строка.
3. В конец выходного значения добавляется двоеточие (:).
4. В конец выходного значения добавляется представление input_bytes в формате base64 в соответствии с разделом 4 [RFC4648] с учётом приведённых ниже требований.
5. В конец выходного значения добавляется двоеточие (:).
6. Возвращается выходная строка.

Закодированные данные (п. 4) дополняются знаками равенства (=), как указано в параграфе 3.2 [RFC4648]. В соответствии с параграфом 3.5 [RFC4648] в кодированных данных также **следует** использовать биты заполнения 0.

4.1.9. Boolean

Для логического значения input_boolean возвращается строка ASCII пригодная для использования в поле HTTP.

1. Если input_boolean не является логическим значением, преобразование завершается отказом.
2. В качестве выходного значения принимается пустая строка.
3. В конец выходного значения добавляется знак вопроса (?).
4. Если input_boolean = true, в конец выходного значения добавляется 1.
5. Если input_boolean = false, в конец выходного значения добавляется 0.
6. Возвращается выходная строка.

4.1.10. Date

Для даты input_date возвращается строка ASCII пригодная для использования в поле HTTP.

1. В качестве выходного значения принимается @.
2. В конец выходного значения добавляется результат преобразования целого числа input_date (параграф 4.1.4).
3. Возвращается выходная строка.

4.1.11. Display String

Для последовательности символов Unicode input_sequence возвращается строка ASCII пригодная для использования в поле HTTP.

1. Если input_sequence не является строкой Unicode, преобразование завершается отказом.
2. В качестве byte_array принимается результат кодирования UTF-8 (раздел 3 в [UTF8]) для input_sequence. При отказе кодирования преобразование завершается отказом.
3. В качестве encoded_string принимается строка %DQUOTE (%").
4. Для каждого байта в byte_array выполняются следующие операции:
    1. Для байтов %x25 (%), %x22 (DQUOTE), %x00-1f, %x7f-ff:
        1. В конец encoded_string добавляется символ процента (%).
        2. В качестве encoded_byte принимается результат кодирования base16 (раздел 8 в [RFC4648]) с переводом букв в нижний регистр;
        3. В конец encoded_string добавляется encoded_byte.
    2. В остальных случаях байт декодируется в символ ASCII, который добавляется в конец encoded_string.
5. В конец encoded_string добавляется DQUOTE.
6. Возвращается encoded_string.

Отметим, что [UTF8] запрещает коды между U+D800 и U+DFFF (суррогаты) и при их наличии в input_sequence преобразование завершается отказом.

4.2. Синтаксический анализ структурированных полей

При синтаксическом анализе принимающей стороной полей HTTP, которые заведомо являются структурированными, важно соблюдать осторожность, поскольку имеется ряд граничных случаев, которые могут вызывать проблемы совместимости и даже безопасности. Этот параграф посвящён алгоритмам синтаксического анализа (разбора).

Для массива байтов input_bytes, представляющих поле-значение (пусто при отсутствии поля) и тип поля (field_type - dictionary, list или item), возвращается полученное при разборе значения поля.

1. Массив input_bytes преобразуется в строку ASCII input_string, а отказ преобразования ведёт к отказу разбора.
2. Отбрасываются все символы SP (пробел) в начале input_string.
3. При field_type list выходным значением (output) будет результат анализа List (параграф 4.2.1) для input_string.
4. При field_type dictionary выходным значением будет результат анализа Dictionary (параграф 4.2.1) для input_string.
5. При field_type item выходным значением будет результат анализа Item (параграф 4.2.1) для input_string.
6. Отбрасываются все символы SP (пробел) в начале input_string.
7. Если строка input_string не пуста, разбор завершается отказом.
8. В ином случае возвращается выходное значение.

При генерации input_bytes синтаксический анализатор **должен** объединить все строки поля из одного раздела (заголовок или трейлер), соответствующие имени поля без учёта регистра символов, в один элемент поле-значение через запятые, как указано в параграфе 5.2 [HTTP] для обеспечения корректной обработки всего поля.

Для списков и словарей это обеспечивает корректную конкатенацию всех строк поля, если отдельные элементы структуры данных верхнего уровня были разнесены по строкам. Алгоритмы разбора для обоих типов допускают наличие символов табуляции, поскольку в некоторых реализациях они могут применяться для объединения строк поля.

Элементы String, разбитые по строкам поля, могут приводить к непредсказуемым результатам, поскольку одна или несколько запятых (возможно, с пробелами) станут частью выходной строки анализатора. Поскольку конкатенацию может выполнять восходящий посредник, результаты не контролируются преобразователем и синтаксическим анализатором, даже если оба находятся под единым управлением.

Элементы Token, Integer, Decimal, Byte Sequence не могут разделяться по нескольким строкам поля, поскольку вставленные запятые приведут к отказу синтаксического анализа.

Анализаторы **могут** давать отказ при обработке значения поля, разделённого на строки, если какая-либо из этих строк не разбирается как данное поле. Например, при разборе поля Example-String, заданного как sf-string разрешен отказ при обработке показанных ниже строк.

```
Example-String: "foo
Example-String: bar"
```

Если при разборе возникает отказ, поле **должно** игнорироваться целиком (как будто его не было в разделе) или, как вариант, все сообщение HTTP **должно** считаться некорректно сформированным. Это требование намеренно является строгим для повышения уровня совместимости и безопасности, а в определениях Structured Field не разрешается отступление от этого требования. Отметим, что требование не применяется к реализациям, не разбирающим поле, например, посредники не обязаны вырезать вызывающие отказ поля при пересылке сообщения.

4.2.1. List

Для ASCII-строки input_string возвращается массив кортежей (item_or_inner_list, parameters), извлечённые значения удаляются из input_string.

1. В качестве members принимается пустая строка.
2. Пока input_string не пуста выполняются следующие действия:
    1. в конец members добавляется результат разбора Item или Inner List (параграф 4.2.1.1) для input_string;
    2. отбрасываются все символы OWS в начале input_string;
    3. если строка input_string пуста, возвращается members;
    4. извлекается первый символ и, если это не запятая (,), разбор завершается отказом;
    5. отбрасываются все символы OWS в начале input_string;
    6. если строка input_string пуста, это завершающая запятая и разбор завершается отказом;
3. Структурированных данных нет, возвращается members (пустое значение).

4.2.1.1. Item или Inner List

Для ASCII-строки input_string возвращается кортеж (item_or_inner_list, parameters), где item_or_inner_list может быть простым элементом или массивом (bare_item, parameters), извлечённые значения удаляются из input_string.

1. Если первым символом input_string является открывающая круглая скобка «(», возвращается результат разбора Inner List (параграф 4.2.1.2) для input_string.
2. Возвращается результат разбора Item (параграф 4.2.3) для input_string.

4.2.1.2. Inner List

Для ASCII-строки input_string возвращается кортеж (inner_list, parameters), где inner_list - массив (bare_item, parameters), извлечённые значения удаляются из input_string.

1. Извлекается первый символ input_string и если это не (, разбор завершается отказом.
2. В качестве inner_list принимается пустой массив.
3. Пока строка input_string не пуста, выполняются следующие операции:
    1. Отбрасываются символы SP в начале input_string.
    2. Если первым символом input_string является), выполняются следующие операции:
        1. извлекается первый символ input_string;
        2. в качестве параметров принимается результат разбора Parameters (параграф 4.2.3.2) для input_string;
        3. возвращается кортеж (inner_list, parameters).
    3. В качестве item принимается результат разбора Item (параграф 4.2.3) для input_string.
    4. В конец inner_list добавляется item.
    5. Если первый символ input_string не SP или), разбор завершается отказом.
4. Конец Inner List не найден, разбор завершается отказом.

4.2.2. Dictionary

Для ASCII-строки input_string возвращается упорядоченный набор кортежей (item_or_inner_list, parameters), извлечённые значения удаляются из input_string.

1. В качестве dictionary принимается пустой набор.
2. Пока строка input_string не пуста, выполняются следующие действия:
    1. В качестве this_key принимается результат разбора ключа Key (параграф 4.2.3.3) для input_string.
    2. Если первым символом input_string является =, выполняются следующие операции:
        1. извлекается первый символ input_string;
        2. в качестве member принимается результат разбора Item или Inner List (параграф 4.2.1.1) для input_string.
    3. В ином случае выполняются следующие операции:
        1. в качестве value принимается логическое (Boolean) значение true;
        2. в качестве parameters принимается результат разбора Parameters (параграф 4.2.3.2) для input_string;
        3. в качестве member принимается кортеж (value, parameters).
    4. Если в словаре уже имеется ключ this_key (посимвольное сравнение), его значение заменяется на member.
    5. В ином случае в конец dictionary добавляется ключ this_key со значением member.
    6. Отбрасываются символы OWS в начале input_string.
    7. Если строка input_string пуста, возвращается dictionary.
    8. Извлекается первый символ input_string и если это не запятая (,), разбор завершается отказом.
    9. Отбрасываются символы OWS в начале input_string.
    10. Если строка input_string пуста, это завершающая запятая и разбор завершается отказом.
3. Структурированных данных не найдено, возвращается dictionary (пустое значение).

Отметим, что при дублировании ключей Dictionary игнорируются все экземпляры, кроме последнего.

4.2.3. Item

Для ASCII-строки input_string возвращается кортеж (bare_item, parameters), извлечённые значения удаляются из input_string.

1. В качестве bare_item принимается результат разбора Bare Item (параграф 4.2.3.1) для input_string.
2. В качестве parameters принимается результат разбора Parameters (параграф 4.2.3.2) для input_string.
3. Возвращается кортеж (bare_item, parameters).

4.2.3.1. Простой элемент

Для ASCII-строки input_string возвращается простой элемент, извлечённые значения удаляются из input_string.

1. Если первым символом input_string является знак минуса (-) или DIGIT, возвращается результат разбора Integer или Decimal (параграф 4.2.4) для input_string.
2. Если первым символом input_string является DQUOTE, возвращается результат разбора String (параграф 4.2.5) для input_string.

3. Если первым символом input_string является ALPHA или *, возвращается результат разбора Token (параграф 4.2.6) для input_string.
4. Если первым символом input_string является двоеточие (:), возвращается результат разбора Byte Sequence (параграф 4.2.7) для input_string.
5. Если первым символом input_string является знак вопроса (?), возвращается результат разбора Boolean (параграф 4.2.8) для input_string.
6. Если первым символом input_string является @, возвращается результат разбора Date (параграф 4.2.9) для input_string.
7. Если первым символом input_string является знак процента (%), возвращается результат разбора Display String (параграф 4.2.10) для input_string.
8. Иное означает, что тип элемента не распознан и разбор завершается отказом.

4.2.3.2. Параметры

Для ASCII-строки input_string возвращается упорядоченный набор простых элементов, извлечённые значения удаляются из input_string.

1. В качестве parameters принимается пустой набор.
2. Пока строка input_string не пуста, выполняются следующие действия:
    1. Если первый символ input_string не точка с запятой (;), выход из цикла.
    2. Извлекается символ точки с запятой (;) из начала input_string.
    3. Отбрасываются символы SP в начале input_string.
    4. В качестве param_key принимается результат разбора ключа (параграф 4.2.3.3) для input_string.
    5. В качестве param_key принимается логическое (Boolean) значение true.
    6. Если первым символом input_string является знак равенства (=), выполняются следующие операции:
        1. Извлекается знак равенства (=) из начала input_string.
        2. В качестве param_key принимается результат разбора Bare Item (параграф 4.2.3.1) для input_string.
    7. Если в parameters уже имеется ключ param_key (посимвольное сравнение), его значение заменяется на param_value.
    8. Иначе ключ param_key со значением param_value добавляется в конец parameters.
3. Возвращается значение parameters.

Отметим, что при дублировании ключей параметра игнорируются все экземпляры, кроме последнего.

4.2.3.3. Ключи

Для ASCII-строки input_string возвращается ключ, извлечённые значения удаляются из input_string.

1. Если первый символ input_string не относится к lcalpha или *, разбор завершается отказом.
2. В качестве output_string принимается пустая строка.
3. Пока строка input_string не пуста, выполняются следующие действия:
    1. Если первый символ input_string не относится к lcalpha, DIGIT, «_», «-», «.» или «*», возвращается output_string.
    2. В качестве char принимается результат извлечения первого символа input_string.
    3. Символ char добавляется в конец output_string.
4. Возвращается значение output_string.

4.2.4. Integer u Decimal

Для ASCII-строки input_string возвращается Integer (параграф 3.3.1) или Decimal (параграф 3.3.2), извлечённые значения удаляются из input_string.

1. В качестве type принимается integer.
2. В качестве sign принимается значение 1.
3. В качестве input_number принимается пустая строка.
4. Если первым символом input_string является знак минус (-), он извлекается и для sign устанавливается значение -1.
5. Если строка input_string пуста, это означает пустое значение integer и разбор завершается отказом.
6. Если первый символ input_string не относится к DIGIT, разбор завершается отказом.
7. Пока строка input_string не пуста, выполняются следующие действия:
    1. В качестве char принимается результат извлечения первого символа input_string.
    2. Если char относится к DIGIT, его значение помещается в конец input_number.

3. В остальных случаях при type = integer и char = «.» выполняются следующие действия:
    1. если input_number содержит более 12 символов, разбор завершается отказом;
    2. в остальных случаях char добавляется в конец input_number и устанавливается type = decimal.
4. Иначе char помещается в начало input_string с выходом из цикла.
5. Если type имеет значение integer, а input_number содержит более 15 символов, разбор завершается отказом.
6. Если type имеет значение decimal, а input_number содержит более 15 символов, разбор завершается отказом.

8. Если type имеет значение integer выполняется следующая операция.
    1. В качестве output_number принимается Integer, т. е. результатом разбора input_number является целое число.
9. В иных случаях выполняются следующие операции.
    1. Если последним символом input_number является точка (.), разбор завершается отказом.
    2. Если число символов input_number после точки больше 3, разбор завершается отказом.
    3. В качестве output_number принимается Decimal, т. е. результатом разбора input_number является десятичное число.
10. В качестве output_number принимается конкатенация sign и output_number[1].
11. Возвращается output_number.

4.2.5. String

Для ASCII-строки input_string возвращается String без кавычек, извлечённые значения удаляются из input_string.

1. В качестве output_string принимается пустая строка.
2. Если первый символ input_string не является DQUOTE, разбор завершается отказом.
3. Отбрасывается первый символ input_string.
4. Пока строка input_string не пуста, выполняются следующие действия:
    1. В качестве char принимается результат извлечения первого символа input_string.
    2. Если char является обратной дробной чертой (\) выполняются следующие действия:
        1. если строка input_string пуста, разбор завершается отказом;
        2. в качестве next_char принимается результат извлечения первого символа input_string;
        3. если next_char не DQUOTE и не \, разбор завершается отказом;
        4. в конец output_string добавляется next_char.
    3. Иначе, если char = DQUOTE, возвращается output_string.
    4. Иначе, если char относится к диапазону %x00-1f или %x7f-ff (не VCHAR или SP), разбор завершается отказом.
    5. Иначе в конец output_string добавляется char.
5. При достижении конца input_string без закрывающего символа DQUOTE разбор завершается отказом.

4.2.6. Token

Для ASCII-строки input_string возвращается Token, извлечённые значения удаляются из input_string.

1. Если первый символ input_string не относится к ALPHA или «*», разбор завершается отказом.
2. В качестве output_string принимается пустая строка.
3. Пока строка input_string не пуста, выполняются следующие действия.
    1. Если первый символ input_string не является tchar, двоеточием (:), или «/», возвращается output_string.
    2. В качестве char принимается результат извлечения первого символа input_string.
    3. В конец output_string добавляется char.
4. Возвращается output_string.

4.2.7. Byte Sequence

Для ASCII-строки input_string возвращается Byte Sequence, извлечённые значения удаляются из input_string.

1. Если первый символ input_string не является двоеточием (:), разбор завершается отказом.
2. Отбрасывается первый символ input_string.
3. Если до конца input_string нет двоеточия (:), разбор завершается отказом.

[1] В оригинале «the product of output_number and sign». *Прим. перев.*

4. В качестве b64_content принимается результат извлечения из input_string всех символов до первого двоеточия, не включая его.
5. Извлекается символ двоеточия (:) в начале input_string.
6. Если b64_content содержит символ вне набора ALPHA, DIGIT, «+», «/», «=», разбор завершается отказом.
7. В качестве binary_content принимается результат base64 [RFC4648] для b64_content, дополненный при необходимости (см. требования ниже). При отказе декодирования base64, разбор завершается отказом.
8. Возвращается binary_content.

Некоторые реализации base64 не разрешают отвергать декодированные данные с некорректным заполнением «=» (см. параграф 3.2 в [RFC4648]), поэтому синтаксическому анализатору **не следует** прерывать разбор при отсутствии «=», если он явно не настроен на это. Некоторые реализации base64 не позволяют отвергать декодированные данные с ненулевыми битами заполнения (см. параграф 3.5 в [RFC4648]), поэтому синтаксическому анализатору **не следует** прерывать разбор при наличии таких битов, если он явно не настроен на это.

Эта спецификация не смягчает требований параграфов 3.1 и 3.3 в [RFC4648], поэтому анализатор **должен** прерывать разбор при наличии символов, не входящих в алфавит base64, и символов перевода строки в кодированных данных.

4.2.8. Boolean

Для ASCII-строки input_string возвращается Boolean, извлечённые значения удаляются из input_string.

1. Если первый символ input_string не является знаком вопроса (?), разбор завершается отказом.
2. Отбрасывается первый символ input_string.
3. Если первым символом input_string является «1», символ отбрасывается и возвращается значение true.
4. Если первым символом input_string является «0», символ отбрасывается и возвращается значение false.
5. Соответствующего значения нет, разбор завершается отказом.

4.2.9. Date

Для ASCII-строки input_string возвращается Date, извлечённые значения удаляются из input_string.

1. Если первым символом input_string не является «@», разбор завершается отказом.
2. Отбрасывается первый символ input_string.
3. В качестве output_date принимается результат Integer или Decimal (параграф 4.2.4) для input_string.
4. Если output_date имеет тип Decimal, разбор завершается отказом.
5. Возвращается output_date.

4.2.10. Display String

Для ASCII-строки input_string возвращается последовательность кодов Unicode, извлечённые значения удаляются из input_string.

1. Если первыми двумя символами input_string не являются % и DQUOTE (%"), разбор завершается отказом.
2. Отбрасываются два первых символа input_string.
3. В качестве byte_array принимается пустой массив.
4. Пока строка input_string не пуста, выполняются следующие операции.
    1. В качестве char принимается результат извлечения первого символа input_string.
    2. Если char относится к диапазону %x00-1f или %x7f-ff (т. е. не VCHAR или SP), разбор завершается отказом.
    3. Если char имеет значение %, выполняются следующие операции:
        1. в качестве octet_hex принимается результат извлечения двух символов input_string, а при их отсутствии разбор завершается отказом;
        2. если octet_hex содержит символы, не относящиеся к %x30-39 или %x61-66 (т. е. не цифры и не буквы a-f в нижнем регистре), разбор завершается отказом;
        3. в качестве octet принимается результат декодирования шестнадцатеричного значения octet_hex (раздел 8 of [RFC4648]);
        4. Значение octet добавляется в конец byte_array.
    4. Если char имеет значение DQUOTE выполняются следующие операции:
        1. в качестве unicode_sequence принимается результат декодирования byte_array как строки UTF-8 (раздел 3 в [UTF8]) и при отказе декодирования разбор завершается отказом;
        2. Возвращается значение unicode_sequence.
    5. Иначе, если char не является знаком процента (%) или DQUOTE, выполняются следующие операции:
        1. В качестве byte принимается результат декодирования ASCII для char.
        2. Значение byte добавляется в конец byte_array.

5. При достижении конца input_string без закрывающего символа DQUOTE разбор завершается отказом.

5. Взаимодействие с IANA

Агентство IANA добавило в реестр Hypertext Transfer Protocol (HTTP) Field Name приведённый ниже текст.

Столбец Structured Type указывает тип поля (в соответствии с RFC 9651) - Dictionary, List или Item (при наличии).

Отметим, что имена полей, начинающиеся с символов, отличных от ALPHA и «*», не могут быть представлены как структурированные поля Token и могут быть не совместимы с отображением в значения ссылающихся на них полей.

В реестр добавлен столбец Structured Type с внесением в него указанных в таблице 1 строк для имеющихся записей.

Таблица 1. Существующие поля.

Имя поля	***Структурированный тип***
Accept-CH	List
Cache-Status	List
CDN-Cache-Control	Dictionary
Cross-Origin-Embedder-Policy	Item
Cross-Origin-Embedder-Policy-Report-Only	Item
Cross-Origin-Opener-Policy	Item
Cross-Origin-Opener-Policy-Report-Only	Item
Origin-Agent-Cluster	Item
Priority	Dictionary
Proxy-Status	List

6. Вопросы безопасности

Размер большинства типов, задаваемых структурированными полями, не ограничен и очень большие поля могут стать объектом атак (например, для истощения ресурсов). Большинство реализаций HTTP ограничивает размер отдельных полей для смягчения таких атак.

Стороны, имеющие возможность внедрять новые поля HTTP, могут менять трактовку структурированных полей. В некоторых случаях это может приводить к отказам синтаксического анализа, но невозможно организовать гарантированные отказы во всех таких обстоятельствах.

Тип Display String позволяет передавать любые возможные коды Unicode без очистки, например, невыделенные коды, управляющие коды (включая NUL) или несимвольные коды. Поэтому приложениям, использующим Display String, необходимо рассмотреть такие стратегии, как фильтрация или экранирование недоверенного содержимого перед его отображением (см. [PRECIS] и [UNICODE-SECURITY]).

7. Литература

7.1. Нормативные документы

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[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>.

[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>.

[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>.

[UTF8] 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>.

7.2. Дополнительная литература

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/info/rfc7541>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE Std 754-2019, DOI 10.1109/IEEESTD.2019.8766229, ISBN 978-1-5044-5924-2, July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[PRECIS] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

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

[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>.

[UNICODE-SECURITY] Davis, M. and M. Suignard, "Unicode Security Considerations", Unicode Technical Report #36, 19 September 2014, <https://www.unicode.org/reports/tr36/tr36-15.html>. Latest version available at <https://www.unicode.org/reports/tr36/>.

Приложение A. Ответы на вопросы

A.1. Почему не JSON?

Ранние предложения по структурированным полям основывались на JSON [RFC8259], однако ограничения на применение с полями HTTP требовали от получателей и отправителей дополнительной специальной обработки. Например, в спецификации JSON имеются проблемы, связанные с большими числами и объектами с дубликатами элементов. Рекомендации по решению этих проблем имеются (например, [RFC7493]), но полагаться на них нельзя.

Строки JSON по умолчанию являются строками Unicode, с чем связан ряд проблем совместимости (например, при сравнении). Разработчикам можно посоветовать избегать без необходимости отличного от ASCII содержимого, но реализацию этого трудно обеспечить. Другим примером является способность JSON размещать содержимое на произвольной глубине. Поскольку требуемый для этого объем памяти может оказаться недоступным (например, во встраиваемых устройствах и других системах с ограничениями), требуется ограничить его тем или иным способом. Существующие реализации JSON не имеют таких ограничений и даже при их задании вполне вероятно, что в каком-либо определении поля ограничения придётся нарушить. Широкое распространение JSON осложняет внесение таких ограничений для всех реализаций, часть их не будет соблюдать ограничения, что приведёт к проблемам совместимости.

Если что-то похоже на JSON, у людей будет соблазн применять JSON для сериализации и синтаксического анализа значений полей. Поскольку основной целью структурированных полей является улучшение функциональной совместимости, и упрощение реализации, эти опасения ведут к необходимости применять специальные преобразователи и синтаксические анализаторы. Кроме того, многие разделяют мнение о том, что JSON «не выглядит правильно» в полях HTTP.

Приложение B. Замечанию по реализации

Базовым реализациям этой спецификации следует предоставлять функции верхнего уровня для преобразования (сериализации, параграф 4.1) и разбора (синтаксического анализ, параграф 4.2). Это не обязательно должны быть функции и можно, например, реализовать их как объекты с методами для каждого типа верхнего уровня. Для обеспечения функциональной совместимости важна полнота реализаций и точное соответствие алгоритмам (см. параграф 1.1). Для оказания помощи в этом сообщество поддерживает специальный открытый ресурс <https://github.com/httpwg/structured-field-tests>.

Разработчикам следует принимать во внимание, что словари и параметры являются упорядоченными. Некоторые поля могут не передавать смысл такого порядка, но его все равно следует показывать, чтобы он был доступен приложениям, которым порядок требуется. При реализации следует осознавать различия между типами Token и String. Хотя большинство языков программирования имеет встроенные типы, которые сопоставляются с другими типами, может потребоваться создание специального объекта token или использование параметров функций для гарантированного разделения этих типов.

Алгоритм преобразования (сериализации) задан так, что он не ограничивается строго типами данных, определёнными в разделе 3. Например, тип Decimal предназначен для расширения диапазона входных значений и округления до допустимых значений.

Реализациям разрешается ограничивать размеры структур с учётом определённого для каждого типа минимума. Если структура превышает заданные реализацией ограничения размера, преобразование или разбор завершаются отказом.

Приложение C. ABNF

В этом приложении используется нотация расширенных форм Бэкуса-Наура (Augmented Backus-Naur Form или ABNF) [RFC5234] для иллюстрации ожидаемого синтаксиса структурированных полей. Однако она не может служить для проверки синтаксиса, поскольку не соответствует всем требованиям. Это приложение не является нормативным и при возникновении противоречий между алгоритмами синтаксического анализа и ABNF алгоритмы имеют преимущество.

```
sf-list       = list-member *( OWS "," OWS list-member )
list-member   = sf-item / inner-list

inner-list    = "(" *SP [ sf-item *( 1*SP sf-item ) *SP ] ")"
                parameters

parameters    = *( ";" *SP parameter )
parameter     = param-key [ "=" param-value ]
param-key     = key
key           = ( lcalpha / "*" )
                *( lcalpha / DIGIT / "_" / "-" / "." / "*" )
lcalpha       = %x61-7A ; a-z
param-value   = bare-item

sf-dictionary = dict-member *( OWS "," OWS dict-member )
dict-member   = member-key ( parameters / ( "=" member-value ))
member-key    = key
member-value  = sf-item / inner-list

sf-item   = bare-item parameters
bare-item = sf-integer / sf-decimal / sf-string / sf-token
```

```
              / sf-binary / sf-boolean / sf-date / sf-displaystring

   sf-integer       = ["-"] 1*15DIGIT
   sf-decimal       = ["-"] 1*12DIGIT "." 1*3DIGIT
   sf-string        = DQUOTE *( unescaped / "%" / bs-escaped ) DQUOTE
   sf-token         = ( ALPHA / "*" ) *( tchar / ":" / "/" )
   sf-binary        = ":" base64 ":"
   sf-boolean       = "?" ( "0" / "1" )
   sf-date          = "@" sf-integer
   sf-displaystring = "%" DQUOTE *( unescaped / "\" / pct-encoded )
                      DQUOTE

   base64       = *( ALPHA / DIGIT / "+" / "/" ) *"="

   unescaped    = %x20-21 / %x23-24 / %x26-5B / %x5D-7E
   bs-escaped   = "\" ( DQUOTE / "\" )

   pct-encoded  = "%" lc-hexdig lc-hexdig
   lc-hexdig = DIGIT / %x61-66 ; 0-9, a-f
```

Приложение D. Отличия от RFC 8941

Эта версия спецификации значений структурированных полей для HTTP (Structured Field Values for HTTP) включает некоторые изменения:

- добавлен тип Date (параграф 3.3.7);
- отменена рекомендация использовать ABNF в определениях новых структурированных полей (раздел 2);
- синтаксис ABNF перенесён в Приложение C;
- добавлен столбец Structured Type в реестр Hypertext Transfer Protocol (HTTP) Field Name (раздел 5);
- усовершенствована обработка ошибок при синтаксическом анализе (параграф 4.2);
- добавлен тип Display String (параграф 3.3.8).

Благодарности

Большое спасибо Matthew Kerwin за предметные отклики и внимательное отношение при разработке спецификации.

Спасибо Ian Clelland, Roy Fielding, Anne van Kesteren, Kazuho Oku, Evert Pot, Julian Reschke, Martin Thomson, Mike West, Jeffrey Yasskin за их вклад в работу.

Адреса авторов

Mark Nottingham
Cloudflare
Prahran VIC
Australia
Email: mnot@mnot.net
URI: https://www.mnot.net/

Poul-Henning Kamp
The Varnish Cache Project
Email: phk@varnish-cache.org

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru