RFC 7232

Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests

문서 최종 수정일	2020-05-26
원문 복사일	2020-05-02
번역 및 정리	이병록(roka88)
이메일	roka88.dev@gmail.com

PROPOSED STANDARD
Errata Exist

Internet Engineering Task Force (IETF)
Request for Comments: 7232
Obsoletes: 2616
Category: Standards Track
ISSN: 2070-1721

R. Fielding, Ed.
Adobe
J. Reschke, Ed.
greenbytes
June 2014

Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests

Abstract

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document defines HTTP/1.1 conditional requests, including metadata header fields for indicating state changes, request header fields for making preconditions on such state, and rules for constructing the responses to a conditional request when one or more preconditions evaluate to false.

하이퍼텍스트 전송 프로토콜(HTTP)은 분산, 협업, 하이퍼텍스트 정보 시스템을 위한 상태 비저장 애플리케이션-레벨 프로토콜이다. 본 문서는 상태 변경을 나타내는 메타데이터 헤더 필드, 해당 상태에 대한 전제조건을 만들기 위한 헤더 필드, 하나 이상의 전제조건이 거짓으로 평가될 때 조건부 요청에 대한 응답을 구성하기 위한 규칙 등 HTTP/1.1 조건부 요청을 정의한다.

Status of This Memo

This is an Internet Standards Track document.

이것은 인터넷 표준 추적 문서이다.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

이 문서는 Internet Engineering Task Force(IETF)의 제품이다. 문서는 IETF 공동체의 합의를 나타낸다. 문서는 공개 검토를 받아왔으며 Internet Engineering Starting Group (IESG)에 의해 발행 승인를 받았다. 인터넷 표준의 추가 정보는 RFC 5741 Section 2에서 확인할 수 있다.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7232.

이 문서에 대한 현재 상태 정보는 정오표와 피드백을 어떻게 제공하는 방법은 http://www.rfc-editor.org/info/rfc7232 에서 얻을 수 있다.

Copyright Notice

Table of Contents

1. Introduction

Conditional requests are HTTP requests [RFC7231] that include one or more header fields indicating a precondition to be tested before applying the method semantics to the target resource. This document defines the HTTP/1.1 conditional request mechanisms in terms of the architecture, syntax notation, and conformance criteria defined in [RFC7230].

조건부 요청은 대상 리소스에 메서드 의미론을 적용하기 전에 테스트되기 위한 전제조건을 나타내는 헤더 필드를 하나 이상 포함하는 HTTP 요청 [RFC7231]이다. 본 문서는 [RFC7230]에서 정의한 아키텍처, 구문 표기법 및 적합성 기준의 관점에서 HTTP/1.1 조건부 요청 메커니즘을 정의한다.

Conditional GET requests are the most efficient mechanism for HTTP cache updates [RFC7234]. Conditionals can also be applied to state-changing methods, such as PUT and DELETE, to prevent the “lost update" problem: one client accidentally overwriting the work of another client that has been acting in parallel.

조건부 GET 요청은 HTTP 캐시 갱신을 위한 가장 효율적인 메커니즘이다 [RFC7234]. 또한 "갱신 손실" 문제를 방지하기 위해 PUT 및 DELETE와 같은 상태 변경 메서드에도 조건을 적용할 수 있다. 즉, 한 클라이언트가 병렬로 행동해 온 다른 클라이언트의 작업을 실수로 덮어쓰게 된다.

Conditional request preconditions are based on the state of the target resource as a whole (its current value set) or the state as observed in a previously obtained representation (one value in that set). A resource might have multiple current representations, each with its own observable state. The conditional request mechanisms assume that the mapping of requests to a “selected representation” (Section 3 of [RFC7231]) will be consistent over time if the server intends to take advantage of conditionals. Regardless, if the mapping is inconsistent and the server is unable to select the appropriate representation, then no harm will result when the precondition evaluates to false.

조건부 요청 전제조건은 대상 리소스의 전체 상태(현재 값 집합) 또는 이전에 획득한 표현(그 집합의 하나의 값)에서 관찰된 상태에 기초한다. 리소스는 각각 관측 가능한 상태를 가진 복수의 현재 표현을 가질 수 있다. 조건부 요청 메커니즘은 서버가 조건들을 이용하려는 경우, "선택된 표현"에 요청의 매핑([RFC7231]의 Section 3)은 시간이 지남에 따라 일관된다고 가정한다. 그럼에도 불구하고, 매핑이 일관성이 없고 서버가 적절한 표현을 선택할 수 없는 경우, 전제조건이 거짓으로 평가될 때 어떠한 위해도 발생하지 않을 것이다.

The conditional request preconditions defined by this specification (Section 3) are evaluated when applicable to the recipient (Section 5) according to their order of precedence (Section 6).

본 명세(Section 3)에서 정의한 조건부 요청 전제조건은 우선 순서에 따라 수신자(Section 5)에게 적용 가능한 경우(Section 6) 평가된다.

1.1 Conformance and Error Handling

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

이 문서의 해당 키워드 “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” 들은 RFC2119 에 설명되어 있다.

Conformance criteria and considerations regarding error handling are defined in Section 2.5 of [RFC7230].

불일치 기준 및 오류 처리와 관련된 고려 사항은 [RFC7230]의 Section 2.5에 정의되어 있다.

1.2 Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with a list extension, defined in Section 7 of [RFC7230], that allows for compact definition of comma-separated lists using a '#' operator (similar to how the '*' operator indicates repetition). Appendix B describes rules imported from other documents. Appendix C shows the collected grammar with all list operators expanded to standard ABNF notation.

이 명세는 [RFC7230]의 Section 7에서 정의한 목록 확장이 있는 [RFC5234]의 ugmented Backus-Naur Form (ABNF) 표기법을 사용하여 '#' 연산자를 사용하여 쉼표로 구분된 목록을 컴팩트하게 정의할 수 있다. (‘*' 연산자가 반복을 나타내는 방식과 유사). 부록 B는 다른 문서에서 가져온 규칙을 설명한다. 부록 C는 표준 ABNF 표기법으로 확장된 모든 목록 연산자와 함께 수집된 문법을 보여준다.

2. Validators

This specification defines two forms of metadata that are commonly used to observe resource state and test for preconditions: modification dates (Section 2.2) and opaque entity tags (Section 2.3). Additional metadata that reflects resource state has been defined by various extensions of HTTP, such as Web Distributed Authoring and Versioning (WebDAV, [RFC4918]), that are beyond the scope of this

specification. A resource metadata value is referred to as a "validator" when it is used within a precondition.

이 명세는 일반적으로 리소스 상태를 관찰하고 전제조건을 테스트하기 위해 사용되는 두 가지 형태의 메타데이터, 즉 수정일 (Section 2.2)과 불투명 엔티티 태그 (Section 2.3)를 정의한다. 리소스 상태를 반영하는 추가 메타데이터는 이 명세의 범위를 벗어난 Web Distributed Authoring 및 Versioning (WebDAV, [RFC4918])과 같은 HTTP의 다양한 확장에 의해 정의되었다. 리소스 메타데이터 값은 전제조건 내에서 사용할 때 “validator”라고 한다.

2.1 Weak versus Strong

Validators come in two flavors: strong or weak. Weak validators are easy to generate but are far less useful for comparisons. Strong validators are ideal for comparisons but can be very difficult (and occasionally impossible) to generate efficiently. Rather than impose that all forms of resource adhere to the same strength of validator, HTTP exposes the type of validator in use and imposes restrictions on when weak validators can be used as preconditions.

검증자는 ‘강한’ 과 ‘약한’ 의 두 가지 형태가 있다. 약한 검증자는 생성하기는 쉽지만 비교에는 훨씬 덜 유용하다. 강한 검증자는 비교에 이상적이지만 효율적인 생성이 매우 어려울 수 있다(그리고 때때로 불가능하다). HTTP는 모든 형태의 리소스가 동일한 검증자의 강도를 고수하도록 강요하기보다는 사용 중인 검증자의 유형을 노출하고 약한 검증자를 전제조건으로 사용할 수 있는 경우에 제한을 가한다.

A "strong validator" is representation metadata that changes value whenever a change occurs to the representation data that would be observable in the payload body of a 200 (OK) response to GET.

"강한 검증자"는 GET에 대한 200(OK) 응답의 페이로드 본문에서 관찰할 수 있는 표현 데이터에 변화가 발생할 때마다 값을 변경하는 표현 메타데이터를 말한다.

A strong validator might change for reasons other than a change to the representation data, such as when a semantically significant part of the representation metadata is changed (e.g., Content-Type), but it is in the best interests of the origin server to only change the value when it is necessary to invalidate the stored responses held by remote caches and authoring tools.

강한 검증자는 표현 메타데이터의 의미상 중요한 부분이 변경되는 경우(e.g., Content-Type)와 같이 표현 데이터 변경 이외의 이유로 변경될 수 있지만, 원격 캐시 및 저작 도구에서 보유하고 있는 저장된 응답을 무효화할 필요가 있을 때만 값을 변경하는 것이 원서버의 최선의 이익이다.

Cache entries might persist for arbitrarily long periods, regardless of expiration times. Thus, a cache might attempt to validate an entry using a validator that it obtained in the distant past. A strong validator is unique across all versions of all representations associated with a particular resource over time. However, there is no implication of uniqueness across representations of different resources (i.e., the same strong validator might be in use for representations of multiple resources at the same time and does not imply that those representations are equivalent).

캐시 항목은 만료 시간에 관계없이 임의로 장기간 지속될 수 있다. 따라서 캐시는 먼 옛날에 얻은 검증자를 사용하여 항목의 유효성을 검사하려고 할 수 있다. 강한 검증자는 시간이 지남에 따라 특정 리소스와 관련된 모든 표현의 모든 버전에서 고유하다. 그러나 서로 다른 리소스의 표현에 걸쳐 고유성의 의미는 없다(즉, 동일한 강한 검증자가 동시에 복수의 리소스의 표현에 사용될 수 있으며 그러한 표현이 동등하다는 의미는 아니다).

There are a variety of strong validators used in practice. The best are based on strict revision control, wherein each change to a representation always results in a unique node name and revision identifier being assigned before the representation is made accessible to GET. A collision-resistant hash function applied to the representation data is also sufficient if the data is available prior to the response header fields being sent and the digest does not need to be recalculated every time a validation request is received. However, if a resource has distinct representations that differ only in their metadata, such as might occur with content negotiation over media types that happen to share the same data format, then the origin server needs to incorporate additional information in the validator to distinguish those representations.

실전에 사용되는 다양한 강한 검증자가 있다. 가장 좋은 것은 엄격한 개정 제어에 기초하며, 그 표현이 GET에 접근가능하게 만들어지기 전에, 표현으로 각 변경될 때 마다 항상 고유한 노드 이름과 개정 식별자가 할당된다. 표현 데이터에 적용되는 충돌 방지 해시함수도 응답 헤더 필드가 전송되기 전에 데이터를 사용할 수 있고 유효성 확인 요청이 수신될 때마다 다이제스트를 다시 계산할 필요가 없는 경우 충분하다. 그러나, 동일한 데이터 형식을 공유하는 미디어 타입에 대한 콘텐츠 협상과 같이 메타데이터에만 차이가 있는 고유한 표현을 리소스가 가지고 있는 경우, 원서버는 그러한 표현을 구별하기 위해 검증자에 추가 정보를 통합해야 한다.

In contrast, a "weak validator" is representation metadata that might not change for every change to the representation data. This weakness might be due to limitations in how the value is calculated, such as clock resolution, an inability to ensure uniqueness for all possible representations of the resource, or a desire of the resource owner to group representations by some self-determined set of equivalency rather than unique sequences of data. An origin server SHOULD change a weak entity-tag whenever it considers prior representations to be unacceptable as a substitute for the current representation. In other words, a weak entity-tag ought to change whenever the origin server wants caches to invalidate old responses.

대조적으로 "약한 검증자"는 표현 데이터의 모든 변경에 대해 변경되지 않을 수 있는 표현 메타데이터다. 이러한 약함은, clock resolution(시간 해상도), 리소스의 가능한 모든 표현에 대한 고유성을 보장할 수 없는 능력, 또는 리소스 소유자가 고유한 데이터 시퀀스 보다 자체-결정된 동등한 집합에 의해 표현을 그룹화하려는 갈망, 어떻게 값이 계산되는지에 대한 제한 때문일 수 있다. 원서버는 이전 표현을 현재 표현을 대체하는 것으로 받아들일 수 없다고 간주할 때마다 약한 entity-tag를 변경해야 한다.(SHOULD) 즉, 약한 entity-tag는 원서버가 오래된 응답을 무효화하기 위해 캐시를 원할 때마다 변경되어야 한다.

For example, the representation of a weather report that changes in content every second, based on dynamic measurements, might be grouped into sets of equivalent representations (from the origin server's perspective) with the same weak validator in order to allow cached representations to be valid for a reasonable period of time (perhaps adjusted dynamically based on server load or weather quality). Likewise, a representation's modification time, if defined with only one-second resolution, might be a weak validator if it is possible for the representation to be modified twice during a single second and retrieved between those modifications.

예를 들어, 동적 측정에 기초하여 매 초마다 내용이 변화하는 기상 보고서의 표현을 캐시된 표현이 합리적인 기간(서버 부하 또는 날씨 품질에 따라 동적으로 조정됨) 동안 유효하도록 하기 위해 동일한 약한 검증자를 가진 동등한 표현 집합(원서버의 관점에서)으로 분류할 수 있다. 마찬가지로 표현의 수정 시간이, one-second resolution로만 정의되는 경우, 표현을 1초 동안 두 번 수정하고 그 수정 사이에 검색할 수 있다면 표현의 수정 시간은 약한 검증자가 될 수 있다.

Likewise, a validator is weak if it is shared by two or more representations of a given resource at the same time, unless those representations have identical representation data. For example, if the origin server sends the same validator for a representation with a gzip content coding applied as it does for a representation with no content coding, then that validator is weak. However, two simultaneous representations might share the same strong validator if they differ only in the

representation metadata, such as when two different media types are available for the same representation data.

마찬가지로, 검증자는 주어진 리소스의 두 개 이상의 표현과 동시에 공유되는 경우, 해당 표현들이 동일한 표현 데이터를 가지고 있지 않는 한, 약하다. 예를 들어, 원서버가 콘텐츠 코딩이 없는 표현과 마찬가지로 gzip 콘텐츠 코딩이 적용된 표현에 대해 동일한 검증자를 보낸다면, 그 검증자는 약하다. 그러나 두 개의 다른 미디어 타입이 동일한 표현 데이터에 대해 사용 가능한 경우와 같이 표현 메타데이터에서만 다른 경우 두 개의 동시 표현은 동일한 강한 검증자를 공유할 수 있다.

Strong validators are usable for all conditional requests, including cache validation, partial content ranges, and "lost update" avoidance. Weak validators are only usable when the client does not require exact equality with previously obtained representation data, such as when validating a cache entry or limiting a web traversal to recent changes.

강한 검증자는 캐시 검증, 부분 콘텐츠 범위, "갱신 손실" 방지 등 모든 조건부 요청에 사용할 수 있다. 약한 검증자는 클라이언트가 캐시 입력을 검증하거나 웹 트래버설을 최근 변경사항으로 제한할 때와 같이 이전에 획득한 표현 데이터와의 정확한 동일성을 요구하지 않는 경우에만 사용할 수 있다.

2.2 Last-Modified

The "Last-Modified" header field in a response provides a timestamp indicating the date and time at which the origin server believes the selected representation was last modified, as determined at the conclusion of handling the request.

응답의 "Last-Modified" 헤더 필드는 요청 처리 완료 시 결정된 대로 원서버가 선택된 표현이 마지막으로 수정되었다고 믿는 날짜와 시간을 나타내는 타임스탬프를 제공한다.

```
Last-Modified = HTTP-date
```

An example of its use is

Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT

2.2.1 Generation

An origin server SHOULD send Last-Modified for any selected representation for which a last modification date can be reasonably and consistently determined, since its use in conditional requests and evaluating cache freshness ([RFC7234]) results in a substantial reduction of HTTP traffic on the Internet and can be a significant factor in improving service scalability and reliability.

원서버는 조건부 요청 및 캐시 신선도 평가([RFC7234])에 사용되어 인터넷에서 HTTP 트래픽이 상당히 감소하고 서비스 확장성 및 안정성을 향상하는 것에 중요한 요소가 될 수 있으므로, 최종 수정 날짜가 합리적이고 일관성 있게 결정될 수 있는 모든 선택된 표현에 대해 최종 수정본을 보내야 한다.(SHOULD)

A representation is typically the sum of many parts behind the resource interface. The last-modified time would usually be the most recent time that any of those parts were changed. How that value is determined for any given resource is an implementation detail beyond the scope of this specification. What matters to HTTP is how recipients of the Last-Modified header field can use its value to make conditional requests and test the validity of locally cached responses.

표현은 일반적으로 리소스 인터페이스 뒤에 있는 많은 부분의 합이다. 마지막 수정 시간은 대개 그러한 부분들 중 어느 것이든 변경된 가장 최근의 시간일 것이다. 주어진 리소스에 대해 그 값이 어떻게 결정되는가는 이 명세의 범위를 벗어나는 구현 세부사항이다. HTTP에서 중요한 것은 Last-Modified 헤더 필드의 수신자가 조건부 요청을 하고 로컬로 캐시된 응답의 유효성을 테스트하기 위해 그것의 값을 어떻게 사용할 수 있는가 하는 것이다.

An origin server SHOULD obtain the Last-Modified value of the representation as close as possible to the time that it generates the Date field value for its response. This allows a recipient to make an accurate assessment of the representation's

modification time, especially if the representation changes near the time that the response is generated.

원서버는 응답에 대한 Date 필드 값을 생성하는 시간에 가능한 가깝도록 표현 Last-Modified 값을 구해야 한다.(SHOULD) 이를 통해 수신자는 특히 응답이 생성되는 시점에 표현이 변경되는 경우, 표현의 수정 시간을 정확하게 평가할 수 있다.

An origin server with a clock MUST NOT send a Last-Modified date that is later than the server's time of message origination (Date). If the last modification time is derived from implementation-specific metadata that evaluates to some time in the future, according to the origin server's clock, then the origin server MUST replace that value with the message origination date. This prevents a future modification date from having an adverse impact on cache validation.

클럭이 있는 원서버는 서버의 메시지 시작 시간 (Date)보다 늦은 Last-Modified 날짜를 보내서는 안 된다.(MUST NOT) 마지막 수정 시간이 원서버의 클럭에 따라 미래의 어느 정도까지 평가하는 구현별 메타데이터에서 파생되는 경우, 원서버는 해당 값을 메시지 발생 날짜로 대체해야 한다.(MUST) 이는 향후 수정 날짜가 캐시 검증에 악영향을 미치지 않도록 방지한다.

An origin server without a clock MUST NOT assign Last-Modified values to a response unless these values were associated with the resource by some other system or user with a reliable clock.

클럭이 없는 원서버는 이 값이 신뢰할 수 있는 클럭을 가진 다른 시스템 또는 사용자에 의해 리소스와 연관되지 않는 한 응답에 Last-Modified 값을 할당해서는 안 된다.(MUST NOT)

2.2.2 Comparison

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

요청에서 유효성 검사자로 사용되는 Last-Modified 시간은 다음과 같은 규칙을 사용하여 강함을 추론할 수 없는 한 암시적으로 약하다.

o The validator is being compared by an origin server to the actual current validator for the representation and,

o 검증자를 표현하기 위해 원서버에서 실제 현재 검증자와 비교하고 있으며,

o That origin server reliably knows that the associated representation did not change twice during the second covered by the presented validator.

o 해당 원서버는 제시된 검증자가 다루는 그 초 동안 관련 표현이 두 번 변경되지 않았음을 신뢰성 있게 알고 있다.

or

또는

o The validator is about to be used by a client in an If-Modified-Since, If-Unmodified-Since, or If-Range header field, because the client has a cache entry for the associated representation, and

o 클라이언트가 관련 표현에 대한 캐시 항목을 가지고 있기 때문에, 검증자는 If-Modified-Since, If-Umodified-Since 또는 If-Range 헤더 필드에서 클라이언트가 사용하려고 함

o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

o 해당 캐시 항목에는 원서버가 원본 응답을 보낸 시간을 제공하는 Date 값이 포함되며

o The presented Last-Modified time is at least 60 seconds before the Date value.

o 표시된 Last-Modified 시간은 Date 값보다 최소 60초 전이다.

or

또는

o The validator is being compared by an intermediate cache to the validator stored in its cache entry for the representation, and

o 중간 캐시에 의해 표현하기 위해 캐시 항목에 저장된 검증자에 대해 검증자를 비교하고 있으며,

o That cache entry includes a Date value, which gives the time when the origin server sent the original response, and

o 해당 캐시 항목에는 원서버가 원래 응답을 보낸 시간을 제공하는 Date 값이 포함되며

o The presented Last-Modified time is at least 60 seconds before the Date value.

o 표시된 최종 수정 시간은 날짜 값보다 최소 60초 전이다.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks or at somewhat different times during the preparation of the response. An implementation MAY use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

이 메서드는 두 개의 다른 응답이 동일한 초 동안 원서버에 의해 전송되었지만 둘 다 마지막 수정 시간이 같다면, 해당 응답 중 적어도 하나는 Last-Modified 시간과 동일한 Date 값을 가질 것이라는 사실에 의존한다. 임의의 60초 제한은 응답 준비 중 다른 시계 또는 다소 다른 시간에 날짜 및 Last-Modified 값이 생성될 가능성을 보호한다. 60초가 너무 짧다고 판단되는 경우, 구현은 60초보다 큰 값을 사용할 것이다.(MAY)

2.3 ETag

The "ETag" header field in a response provides the current entity-tag for the selected representation, as determined at the conclusion of handling the request. An entity-tag is an opaque validator for differentiating between multiple representations of the same resource, regardless of whether those multiple representations are due to resource state changes over time, content negotiation resulting in multiple representations being valid at the same time, or both. An entity-tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

응답의 "ETag" 헤더 필드는 요청 처리 완료 시 결정된 대로 선택된 표현에 대한 현재 entity-tag를 제공한다. entity-tag는 복수의 표현이 시간에 따른 리소스 상태 변화, 복수의 표현을 동시에 유효하게 만드는 콘텐츠 협상, 또는 둘 다에 관계없이 동일한 리소스의 복수의 표현을 구별하기 위한 불투명한 검증자다. entity-tag는 불투명한 따옴표 문자열로 구성되며, 약한 표시기에 의해 접두사가 될 수 있다.

```
ETag       = entity-tag

entity-tag = [ weak ] opaque-tag
weak       = %x57.2F ; "W/", case-sensitive
opaque-tag = DQUOTE *etagc DQUOTE
etagc      = %x21 / %x23-7E / obs-text
           ; VCHAR except double quotes, plus obs-text
```

> Note: Previously, opaque-tag was defined to be a quoted-string ([RFC2616], Section 3.11); thus, some recipients might perform backslash unescaping. Servers therefore ought to avoid backslash characters in entity tags.
>
> 참고: 이전에는 불투명 태그가 따옴표 문자열([RFC2616], Section 3.11)로 정의되었기 때문에 일부 수신자는 백슬래시 언 이스케이핑을 수행할 수 있다. 따라서 서버는 엔티티 태그의 문자를 역슬래시하지 않아야 한다.

An entity-tag can be more reliable for validation than a modification date in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where modification dates are not consistently maintained.

변경일을 저장하는 것이 불편하거나 HTTP 날짜 값의 one-second resolution이 충분하지 않거나 변경일이 일관되게 유지되지 않는 상황에서는 변경일보다 entity-tag를 더 신뢰할 수 있다.

Examples:

예시로:

ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""

An entity-tag can be either a weak or strong validator, with strong being the default. If an origin server provides an entity-tag for a representation and the generation of that entity-tag does not satisfy all of the characteristics of a strong validator (Section 2.1), then the origin server MUST mark the entity-tag as weak by prefixing its opaque value with "W/" (case-sensitive).

entity-tag는 약한 검증자 또는 강한 검증자가 될 수 있으며, 강한 검증자는 기본값이다. 원서버가 표현에 대한 entity-tag를 제공하고 해당 entity-tag의 생성이 강한 검증자의 모든 특성을 만족시키지 못하는 경우(Section 2.1), 원서버는 불투명 값에 "W/"(대소문자 구분) 접두사를 붙여 entity-tag를 약함으로 표시해야 한다.(MUST)

2.3.1 Generation

The principle behind entity-tags is that only the service author knows the implementation of a resource well enough to select the most accurate and efficient validation mechanism for that resource, and that any such mechanism can be mapped to a simple sequence of octets for easy comparison. Since the value is opaque, there is no need for the client to be aware of how each entity-tag is constructed.

entity-tag의 원칙은 서비스 저자만이 해당 리소스에 대한 가장 정확하고 효율적인 검증 메커니즘을 선택할 수 있을 만큼 리소스의 구현을 잘 알고 있으며, 그러한 메커니즘은 비교가 쉽도록 단순한 octet 시퀀스에 매핑될 수 있다는 것이다. 값이 불투명하기 때문에, 클라이언트는 각 entity-tag가 어떻게 구성되는지 알 필요가 없다.

For example, a resource that has implementation-specific versioning applied to all changes might use an internal revision number, perhaps combined with a variance identifier for content negotiation, to accurately differentiate between representations. Other implementations might use a collision-resistant hash of representation content, a combination of various file attributes, or a modification timestamp that has sub-second resolution.

예를 들어, 모든 변경에 구현별 버전 지정을 적용한 리소스는 표현을 정확하게 구분하기 위해 내부 개정 번호(콘텐츠 협상을 위한 분산 식별자)와 결합할 수 있다. 다른 구현에서는 충돌 방지 표현 콘텐츠의 해시, 다양한 파일 속성의 조합 또는 sub-second resolution를 가진 수정 타임스탬프를 사용할 수 있다.

An origin server SHOULD send an ETag for any selected representation for which detection of changes can be reasonably and consistently determined, since the entity-tag's use in conditional requests and evaluating cache freshness ([RFC7234]) can result in a substantial reduction of HTTP network traffic and can be a significant factor in improving service scalability and reliability.

원서버는 조건부 요청 및 캐시 신선도 평가([RFC7234])에서 entity-tag의 사용이 HTTP 트래픽이 상당히 감소하고 서비스 확장성 및 안정성을 향상하는 것에 중요한 요소가 될 수 있으므로, 변경사항의 탐지를 합리적이고 일관성 있게 결정할 수 있는 선택된 표현에 대해 ETag를 보내야 한다.(SHOULD)

2.3.2 Comparison

There are two entity-tag comparison functions, depending on whether or not the comparison context allows the use of weak validators:

비교 컨텍스트에서 약한 검증자를 사용할 수 있는지 여부에 따라 두 가지 entity-tag 비교 함수가 있다.

o Strong comparison: two entity-tags are equivalent if both are not weak and their opaque-tags match character-by-character.

o 강한 비교: 두 개의 entity-tag가 모두 약하지 않고 opaque-tag가 문자별로 일치할 경우 동일하다.

o Weak comparison: two entity-tags are equivalent if their opaque-tags match character-by-character, regardless of either or both being tagged as "weak".

o 약한 비교: 두 개의 entity-tag는 "약한" 태그 중 하나 또는 둘 다에 상관없이 문자별 태그와 일치하는 경우 동등하다.

The example below shows the results for a set of entity-tag pairs and both the weak and strong comparison function results:

아래 예제에서는 엔티티 태그 쌍 집합과 약한 및 강한 비교 함수의 결과를 모두 보여 준다.

ETag 1	ETag 2	Strong Comparison	Weak Comparison
W/"1"	W/"1"	no match	match
W/"1"	W/"2"	no match	no match
W/"1"	"1"	no match	match
"1"	"1"	match	match

2.3.3 Example: Entity-Tags Varying on Content-Negotiated Resources

Consider a resource that is subject to content negotiation (Section 3.4 of [RFC7231]), and where the representations sent in response to a GET request vary based on the Accept-Encoding request header field (Section 5.3.4 of [RFC7231]):

콘텐츠 협상 대상([RFC7231]의 Section 3.4)이며 GET 요청에 응답하여 전송된 표현은 Accept-Encoding 요청 헤더 필드([RFC7231]의 Section 5.3.4)에 따라 달라질 수 있는 리소스를 고려하면:

>> Request:

```
GET /index HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip
```

In this case, the response might or might not use the gzip content coding. If it does not, the response might look like:

이 경우, 응답은 gzip 콘텐츠 코딩을 사용할 수도 있고 사용하지 않을 수도 있다. 그렇지 않으면 다음과 같이 응답할 수 있다.

>> Response:

HTTP/1.1 200 OK
Date: Fri, 26 Mar 2010 00:05:00 GMT
ETag: "123-a"
Content-Length: 70
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
Hello World!
Hello World!
Hello World!
Hello World!

An alternative representation that does use gzip content coding would be:

gzip 콘텐츠 코딩을 사용하는 대안적 표현은

>> Response:

HTTP/1.1 200 OK
Date: Fri, 26 Mar 2010 00:05:00 GMT
ETag: "123-b"
Content-Length: 43
Vary: Accept-Encoding
Content-Type: text/plain
Content-Encoding: gzip

...binary data...

Note: Content codings are a property of the representation data, so a strong entity-tag for a content-encoded representation has to be distinct from the entity

tag of an unencoded representation to prevent potential conflicts during cache updates and range requests. In contrast, transfer codings (Section 4 of [RFC7230]) apply only during message transfer and do not result in distinct entity-tags.

참고: 콘텐츠 코딩은 표현 데이터의 속성이므로, 콘텐츠 인코딩된 표현을 위한 강한 entity-tag는 캐시 갱신 및 범위 요청 중 잠재적 충돌을 방지하기 위해 인코딩되지 않은 표현의 엔티티 태그와 구별되어야 한다. 대조적으로, 전송 코딩([RFC7230]의 Section 4)은 메시지 전송 중에만 적용되며 별개의 entity-tag가 발생하지 않는다.

2.4. When to Use Entity-Tags and Last-Modified Dates

In 200 (OK) responses to GET or HEAD, an origin server:

GET 또는 HEAD에 대한 200(확인) 응답에서 원서버:

o SHOULD send an entity-tag validator unless it is not feasible to generate one.

o entity-tag 검증자를 생성할 수 없는 경우가 아니라면 보내야 한다.(SHOULD)

o MAY send a weak entity-tag instead of a strong entity-tag, if performance considerations support the use of weak entity-tags, or if it is unfeasible to send a strong entity-tag.

o 성능 고려사항이 약한 entity-tag의 사용을 뒷받침하거나 강한 entity-tag를 보내는 것이 불가능할 경우, 강한 entity-tag 대신 약한 entity-tag를 보낼 수 있다.

o SHOULD send a Last-Modified value if it is feasible to send one.

o Last-Modified 값을 보낼 수 있다면 보내야 한다.(SHOULD)

In other words, the preferred behavior for an origin server is to send both a strong entity-tag and a Last-Modified value in successful responses to a retrieval request.

즉, 원서버에서 선호하는 동작은 검색 요청에 대한 성공적인 응답에서 강한 entity-tag와 Last-Modified 값을 모두 보내는 것이다.

A client:

클라이언트:

o MUST send that entity-tag in any cache validation request (using If-Match or If-None-Match) if an entity-tag has been provided by the origin server.

o 원서버에서 entity-tag를 제공한 경우 캐시 유효성 검사 요청(If-Match 또는 If-None-Match 사용)에서 entity-tag를 전송해야 한다.(MUST)

o SHOULD send the Last-Modified value in non-subrange cache validation requests (using If-Modified-Since) if only a Last-Modified value has been provided by the origin server.

o 원서버에서 Last-Modified 값만 제공한 경우, 비 하위 범위 캐시 유효성 검사 요청(If-Modified-Since 사용)에서 Last-Modified 값을 전송해야 한다.(SHOULD)

o MAY send the Last-Modified value in subrange cache validation requests (using If-Unmodified-Since) if only a Last-Modified value has been provided by an HTTP/1.0 origin server. The user agent SHOULD provide a way to disable this, in case of difficulty.

o HTTP/1.0 원서버에서 Last-Modified 값만 제공한 경우 하위 범위 캐시 유효성 검사 요청(If-Unmodified-Since 사용)에서 Last-Modified 값을 전송할 수 있다.(MAY) 어려울 경우, 사용자 에이전트는 이를 비활성화할 수 있는 방법을 제공해야 한다.(SHOULD)

o SHOULD send both validators in cache validation requests if both an entity-tag and a Last-Modified value have been provided by the origin server. This allows both HTTP/1.0 and HTTP/1.1 caches to respond appropriately.

o entity-tag와 Last-Modified 값이 모두 원서버에 의해 제공된 경우 캐시 유효성 검사 요청에서 두 검증자를 보내야 한다. 이를 통해 HTTP/1.0 및 HTTP/1.1 캐시가 모두 적절하게 응답할 수 있다.

3. Precondition Header Fields

This section defines the syntax and semantics of HTTP/1.1 header fields for applying preconditions on requests. Section 5 defines when the preconditions are applied. Section 6 defines the order of evaluation when more than one precondition is present.

이 섹션에서는 요청에 전제조건을 적용하기 위한 HTTP/1.1 헤더 필드의 구문과 의미론을 정의한다. Section 5는 전제조건이 적용되는 시기를 정의한다. Section 6은 둘 이상의 전제조건이 존재할 때의 평가 순서를 정의한다.

3.1. If-Match

The "If-Match" header field makes the request method conditional on the recipient origin server either having at least one current representation of the target resource, when the field-value is "*", or having a current representation of the target resource that has an entity-tag matching a member of the list of entity-tags provided in the field-value.

"If-Match" 헤더 필드는 적어도 하나 이상의 대상 리소스의 현재 표현을 가지거나, field-value가 "*"일 때, 또는 field-value에 제공된 entity-tag 목록의 멤버와 일치하는 entity-tag가 있는 대상 리소스를 현재 표현을 가지고 있을 때, 수신자 원서버에서 요청 메서드를 조건적으로 만든다.

An origin server MUST use the strong comparison function when comparing entity-tags for If-Match (Section 2.3.2), since the client intends this precondition to prevent the method from being applied if there have been any changes to the representation data.

원서버는 If-Match에 대한 entity-tag(Section 2.3.2)를 비교할 때 강한 비교 함수를 사용해야 한다.(MUST) 클라이언트는 표현 데이터에 변경이 있는 경우 그 메서드를 적용하는 것을 방지하기 위해 이 전제조건을 충족해야 한다.

```
If-Match = "*" / 1#entity-tag
```

Examples:

예시로:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

If-Match is most often used with state-changing methods (e.g., POST, PUT, DELETE) to prevent accidental overwrites when multiple user agents might be acting in parallel on the same resource (i.e., to prevent the "lost update" problem). It can also be used with safe methods to abort a request if the selected representation does not match one already stored (or partially stored) from a prior request.

If-Match는 복수의 사용자 에이전트가 동일한 리소스를 병렬로 실행할 때(i.e., "갱신 손실" 문제를 방지하기 위해) 우발적인 덮어쓰기를 방지하기 위해 상태 변경 메서드(e.g., POST, PUT, DELETE)에 가장 많이 사용된다. 또한 선택된 표현이 이전 요청에서 이미 저장(또는 부분적으로 저장)된 것과 일치하지 않을 경우 안전 메서드와 요청을 중단하기 위해 사용될 수 있다.

An origin server that receives an If-Match header field MUST evaluate the condition prior to performing the method (Section 5). If the field-value is "*", the condition is false if the origin server does not have a current representation for the target resource. If the field-value is a list of entity-tags, the condition is false if none of the listed tags match the entity-tag of the selected representation.

If-Match 헤더 필드를 수신하는 원서버는 그 메서드를 수행하기 전에 조건을 평가해야 한다.(MUST) (Section 5) field-value가 "*"인 경우, 원서버에 대상 리소스에 대한 현재 표시가 없는 경우 조건은 거짓이다. field-value가 entity-tag 목록인 경우, 나열된 태그 중 선택된 표현의 entity-tag와 일치하는 태그가 없는 경우 조건은 거짓이다.

An origin server MUST NOT perform the requested method if a received If-Match condition evaluates to false; instead, the origin server MUST respond with either a)

the 412 (Precondition Failed) status code or b) one of the 2xx (Successful) status codes if the origin server has verified that a state change is being requested and the final state is already reflected in the current state of the target resource (i.e., the change requested by the user agent has already succeeded, but the user agent might not be aware of it, perhaps because the prior response was lost or a compatible change was made by some other user agent). In the latter case, the origin server MUST NOT send a validator header field in the response unless it can verify that the request is a duplicate of an immediately prior change made by the same user agent.

수신된 If-Match 조건이 거짓으로 평가될 경우 원서버는 요청된 메서드를 수행하지 않아야 하며, 대신 원서버는
a) 412(Precondition Failed) 상태 코드 또는
b) 상태 변경이 요청되는 중이며 대상 리소스의 현재 상태에서 최종 상태가 이미 반영된 것을 원서버가 확인하는 경우 2xx(Successful) 상태 코드 중 하나로 응답해야 한다.(MUST)
(i.e., 사용자 에이전트에서 요청한 변경은 이미 성공했지만, 사전 응답이 손실되었거나 다른 사용자 에이전트에 의해 호환 가능한 변경이 이루어졌기 때문에 사용자 에이전트는 이를 알지 못할 수 있다).
후자의 경우, 원서버는 요청이 동일한 사용자 에이전트에 의해 이루어진 즉시 이전 변경사항의 중복인지 확인할 수 없는 한, 응답에 있는 검증자 헤더 필드를 전송해서는 안 된다.(MUST NOT)

The If-Match header field can be ignored by caches and intermediaries because it is not applicable to a stored response.

If-Match 헤더 필드는 저장된 응답에 적용되지 않기 때문에 캐시와 중개자에 의해 무시될 수 있다.

3.2. If-None-Match

The "If-None-Match" header field makes the request method conditional on a recipient cache or origin server either not having any current representation of the target resource, when the field-value is "*", or having a selected representation with an entity-tag that does not match any of those listed in the field-value.

"If-None-Match" 헤더 필드는 대상 리소스에 대한 어느 하나 현재 표현을 가지고 있지 않거나, 필드 값이 "*" 일 때, 또는 field-value에 나열된 entity-tag 목록의 멤버와 어느 하나 일치하지 않는 entity-tag로 선택된 표현을 가지고 있지 않을 때, 수신자 캐시 또는 원서버에서 요청 메서드를 조건적으로 만든다.

A recipient MUST use the weak comparison function when comparing entity-tags for If-None-Match (Section 2.3.2), since weak entity-tags can be used for cache validation even if there have been changes to the representation data.

수신자는 If-None-Match에 대한 entity-tag를 비교할 때 약한 비교 함수를 사용해야 한다.(MUST) (Section 2.3.2) 표현 데이터가 변경된 경우에도 약한 entity-tag를 캐시 유효성 검사에 사용할 수 있기 때문이다.

```
If-None-Match = "*" / 1#entity-tag
```

Examples:

예시로:

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

If-None-Match is primarily used in conditional GET requests to enable efficient updates of cached information with a minimum amount of transaction overhead. When a client desires to update one or more stored responses that have entity-tags, the client SHOULD generate an If-None-Match header field containing a list of those entity-tags when making a GET request; this allows recipient servers to send a 304 (Not Modified) response to indicate when one of those stored responses matches the selected representation.

If-None-Match는 최소한의 트랜잭션 오버헤드로 캐시된 정보를 효율적으로 갱신할 수 있도록 조건부 GET 요청에 주로 사용된다. 클라이언트가 entity-tag를 가진 하나 이상의 저장된 응답을 갱신하고자 할 때, 클라이언트는 GET 요청 시 해당 entity-tag의 목록이 포함된 If-None-Match 헤더 필드를 생성해야 한다.(SHOULD) 이렇게 하면 수신자 서버가 304(Not

Modified) 응답을 보내 해당 저장된 응답 중 하나가 선택된 표현과 일치할 때를 나타낼 수 있다.

If-None-Match can also be used with a value of "*" to prevent an unsafe request method (e.g., PUT) from inadvertently modifying an existing representation of the target resource when the client believes that the resource does not have a current representation (Section 4.2.1 of [RFC7231]). This is a variation on the "lost update" problem that might arise if more than one client attempts to create an initial representation for the target resource.

클라이언트는 리소스가 현재 표현을 가지고 있지 않다고 판단할 때, If-None-Match로 "*" 값을 사용하여 안전하지 않은 요청 메서드(e.g., PUT)의 대상 리소스의 기존 표현을 실수로 수정하지 못하도록 할 수 있다([RFC7231]의 Section 4.2.1). 이는 둘 이상의 클라이언트가 대상 리소스에 대한 초기 표현을 작성하려고 시도할 경우 발생할 수 있는 "갱신 손실" 문제에 대한 변화이다.

An origin server that receives an If-None-Match header field MUST evaluate the condition prior to performing the method (Section 5). If the field-value is "*", the condition is false if the origin server has a current representation for the target resource. If the field-value is a list of entity-tags, the condition is false if one of the listed tags match the entity-tag of the selected representation.

If-None-Match 헤더 필드를 수신하는 원서버는 메서드를 수행하기 전에 조건을 평가해야 한다.(MUST) (Section 5) field-value가 "*"인 경우, 원서버가 대상 리소스에 대한 현재 표현을 가지고 있는 경우 조건은 거짓이다. field-value가 entity-tag의 목록인 경우, 나열된 태그 중 하나가 선택된 표현의 entity-tag와 일치할 경우 그 조건은 거짓이다.

An origin server MUST NOT perform the requested method if the condition evaluates to false; instead, the origin server MUST respond with either a) the 304 (Not Modified) status code if the request method is GET or HEAD or b) the 412 (Precondition Failed) status code for all other request methods.

원서버는 조건이 거짓으로 평가될 경우 요청된 메서드를 수행해서는 안 되며(MUST NOT);대신,
a) 요청 메서드가 GET 또는 HEAD이면 304 (Not Modified) 상태 코드로 또는
b) 기타 모든 요청 메서드에 대해 412 (Precondition Failed) 상태 코드로 응답해야 한다.

Requirements on cache handling of a received If-None-Match header field are defined in Section 4.3.2 of [RFC7234].

수신된 If-None-Match 헤더 필드의 캐시 처리 요건은 [RFC7234]의 Section 4.3.2에 정의되어 있다.

3.3. If-Modified-Since

The "If-Modified-Since" header field makes a GET or HEAD request method conditional on the selected representation's modification date being more recent than the date provided in the field-value. Transfer of the selected representation's data is avoided if that data has not changed.

"If-Modified-Since" 헤더 필드는 선택된 표현의 수정일이 field-value에 제공된 날짜보다 더 최신인 것을 조건으로 GET 또는 HEAD 요청 메서드를 만든다. 데이터가 변경되지 않은 경우 선택된 표현 데이터의 전송을 피한다.

```
If-Modified-Since = HTTP-date
```

An example of the field is:

필드의 예로:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A recipient MUST ignore If-Modified-Since if the request contains an If-None-Match header field; the condition in If-None-Match is considered to be a more accurate replacement for the condition in If-Modified-Since, and the two are only combined for the sake of interoperating with older intermediaries that might not implement If-None-Match.

수신자는 요청이 If-None-Match 헤더 필드를 포함한다면 If-Modified-Since를 반드시 무시해야 한다.(MUST) If-None-Match의 조건은 If-Modified-Since 조건 보다 더 정확한 대체물로

간주되며, 두 조건은 If-None-Match를 구현하지 않을 수 있는 이전 중개자와의 상호 운용성을 위해서만 결합된다.

A recipient MUST ignore the If-Modified-Since header field if the received field-value is not a valid HTTP-date, or if the request method is neither GET nor HEAD.

수신자는 수신된 field-value가 유효한 HTTP-date가 아니거나, 요청 메서드가 GET 또는 HEAD가 아닌 경우 If-Modified-Since 헤더 필드를 반드시 무시해야 한다.(MUST)

A recipient MUST interpret an If-Modified-Since field-value's timestamp in terms of the origin server's clock.

수신자는 원서버의 클럭으로 If-Modified-Since field-value의 타임스탬프를 반드시 해석해야 한다.(MUST)

If-Modified-Since is typically used for two distinct purposes: 1) to allow efficient updates of a cached representation that does not have an entity-tag and 2) to limit the scope of a web traversal to resources that have recently changed.

If-Modified-Since는 일반적으로 두 가지의 구분되는 목적으로 사용된다.
1) entity-tag 없는 캐시된 표현을 효율적으로 갱신할 수 있도록 허용하고
2) 웹 트래버설의 범위를 최근에 변경된 리소스로 제한한다.

When used for cache updates, a cache will typically use the value of the cached message's Last-Modified field to generate the field value of If-Modified-Since. This behavior is most interoperable for cases where clocks are poorly synchronized or when the server has chosen to only honor exact timestamp matches (due to a problem with Last-Modified dates that appear to go "back in time" when the origin server's clock is corrected or a representation is restored from an archived backup). However, caches occasionally generate the field value based on other data, such as the Date header field of the cached message or the local clock time that the message was received, particularly when the cached message does not contain a Last-Modified field.

캐시 갱신에 사용할 경우, 캐시는 일반적으로 캐시된 메시지의 Last-Modified 필드 값을 사용하여 If-Modified-Since의 필드 값을 생성한다. 이 동작은 시계가 제대로 동기화되지 않았거나 서버가 정확한 타임스탬프 일치만 준수하도록 선택한 경우(원서버의 시계를 수정하거나 보관된 백업에서 표현이 복원될 때 "시간 되돌리기"로 보이는 마지막 수정 날짜의 문제 때문에)에 대해 가장 상호운용 가능하다. 그러나 캐시는 캐시된 메시지의 Date 헤더 필드 또는 메시

지가 수신된 로컬 시계 시간과 같은 다른 데이터에 기반하여 필드 값을 생성하는 경우가 있으며, 특히 캐시된 메시지에 Last-Modified 필드가 포함되어 있지 않은 경우이다.

When used for limiting the scope of retrieval to a recent time window, a user agent will generate an If-Modified-Since field value based on either its own local clock or a Date header field received from the server in a prior response. Origin servers that choose an exact timestamp match based on the selected representation's Last-Modified field will not be able to help the user agent limit its data transfers to only those changed during the specified window.

검색 범위를 최근 시간대로 제한하기 위해 사용할 경우, 사용자 에이전트는 자체 로컬 시계 또는 사전 응답으로 서버로부터 수신한 Date 헤더 필드에 기반한 If-Modified-Since 필드 값을 생성한다. 선택된 표현의 Last-Modified 필드를 기준으로 정확한 타임스탬프 일치를 선택하는 원서버는 사용자 에이전트가 지정된 기간 동안 변경된 데이터로만 데이터 전송을 제한하는 것을 시킬 수 없을 것이다.

An origin server that receives an If-Modified-Since header field SHOULD evaluate the condition prior to performing the method (Section 5). The origin server SHOULD NOT perform the requested method if the selected representation's last modification date is earlier than or equal to the date provided in the field-value; instead, the origin server SHOULD generate a 304 (Not Modified) response, including only those metadata that are useful for identifying or updating a previously cached response.

If-Modified-Since 헤더 필드를 수신하는 원서버는 메서드를 수행하기 전에 조건을 평가해야 한다.(SHOULD) (Section 5) 선택된 표현의 마지막 수정 날짜가 필드 값에 제공된 날짜보다 빠르거나 같으면 원서버가 요청된 메서드를 수행하지 않아야 하며;(SHOULD NOT) 대신, 원서버는 이전에 캐시된 응답을 식별하거나 갱신하는 데 유용한 메타데이터만 포함하여 304 (Not Modified) 응답을 생성해야 한다.(SHOULD)

Requirements on cache handling of a received If-Modified-Since header field are defined in Section 4.3.2 of [RFC7234].

수신된 If-Modified-Since 헤더 필드의 캐시 처리 요건은 [RFC7234]의 Section 4.3.2에 정의되어 있다.

3.4. If-Unmodified-Since

The "If-Unmodified-Since" header field makes the request method conditional on the selected representation's last modification date being earlier than or equal to the date provided in the field-value.

"If-Unmodified-Since" 헤더 필드는 선택된 표현의 마지막 수정일을 field-value에 제공된 날짜보다 이전 또는 같은 날짜로 조건으로 요청 메서드를 만든다.

This field accomplishes the same purpose as If-Match for cases where the user agent does not have an entity-tag for the representation.

이 필드는 사용자 에이전트에 표현에 대한 entity-tag가 없는 경우 If-Match와 동일한 목적을 달성한다.

```
If-Unmodified-Since = HTTP-date
```

An example of the field is:

필드의 예로:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A recipient MUST ignore If-Unmodified-Since if the request contains an If-Match header field; the condition in If-Match is considered to be a more accurate replacement for the condition in If-Unmodified-Since, and the two are only combined for the sake of interoperating with older intermediaries that might not implement If-Match.

수신자는 요청이 If-Match 헤더 필드를 포함한다면 If-Unmodified-Since를 반드시 무시해야 한다.(MUST): If-Match의 조건은 If-Unmodified-Since 조건 보다 더 정확한 대체물로 간주되며, 두 조건은 If-Match를 구현하지 않을 수 있는 이전 중개자와의 상호 운용성을 위해서만 결합된다.

A recipient MUST ignore the If-Unmodified-Since header field if the received field-value is not a valid HTTP-date.

수신자는 수신된 field-value가 유효한 HTTP-date가 아니면 If-Unmodified-Since 헤더 필드를 반드시 무시해야 한다.(MUST)

A recipient MUST interpret an If-Unmodified-Since field-value's timestamp in terms of the origin server's clock.

수신자는 원서버의 클럭으로 If-Unmodified-Since field-value의 타임스탬프를 반드시 해석해야 한다.(MUST)

If-Unmodified-Since is most often used with state-changing methods (e.g., POST, PUT, DELETE) to prevent accidental overwrites when multiple user agents might be acting in parallel on a resource that does not supply entity-tags with its representations (i.e., to prevent the "lost update" problem). It can also be used with safe methods to abort a request if the selected representation does not match one already stored (or partially stored) from a prior request.

If-Unmodified-Since에는 여러 사용자 에이전트가 표현과 함께 entity-tag를 제공하지 않는 리소스(즉, "갱신 손실" 문제를 방지하기 위해)에서 병렬로 작업할 때 우발적인 덮어쓰기를 방지하기 위해 상태 변경 메서드(e.g., POST, PUT, DELETE)에 가장 많이 사용된다. 또한 선택된 표현이 이전 요청에서 이미 저장(또는 부분적으로 저장)된 것과 일치하지 않을 경우 안전한 메서드로 요청을 중단할 수 있다.

An origin server that receives an If-Unmodified-Since header field MUST evaluate the condition prior to performing the method (Section 5). The origin server MUST NOT perform the requested method if the selected representation's last modification date is more recent than the date provided in the field-value; instead the origin server MUST respond with either a) the 412 (Precondition Failed) status code or b) one of the 2xx (Successful) status codes if the origin server has verified that a state change is being requested and the final state is already reflected in the current state of the target resource (i.e., the change requested by the user agent has already succeeded, but the user agent might not be aware of that because the prior response message was lost or a compatible change was made by some other user agent). In the latter case, the origin server MUST NOT send a validator header

field in the response unless it can verify that the request is a duplicate of an immediately prior change made by the same user agent.

If-Unmodified-Since 헤더 필드를 수신하는 원서버는 메소드를 수행하기 전에 조건을 평가해야 한다.(MUST) (Section 5) 선택된 표현의 마지막 수정 날짜가 field-value에 제공된 날짜보다 더 최근의 경우 원서버는 요청된 메서드를 수행하지 않아야 하며, 대신 원서버는
a) 412(Precondition Failed) 상태 코드 또는
b) 상태 변경이 요청되는 중이며 대상 리소스의 현재 상태에서 최종 상태가 이미 반영된 것을 원서버가 확인하는 경우 2xx(Successful) 상태 코드 중 하나로 응답해야 한다.(MUST)
(i.e., 사용자 에이전트에서 요청한 변경은 이미 성공했지만, 사전 응답 메시지가 손실되었거나 다른 사용자 에이전트에서 호환 가능한 변경사항이 발생했기 때문에 사용자 에이전트는 이를 알지 못할 수 있다).
후자의 경우, 원서버는 요청이 동일한 사용자 에이전트에 의해 이루어진 즉시 이전 변경사항의 중복인지 확인할 수 없는 한 응답에 있는 검증자 헤더 필드를 전송해서는 안 된다.(MUST NOT)

The If-Unmodified-Since header field can be ignored by caches and intermediaries because it is not applicable to a stored response.

If-Unmodified-Since 헤더 필드는 저장된 응답에 적용할 수 없기 때문에 캐시 및 중개자에 의해 무시될 수 있다.

3.5. If-Range

The "If-Range" header field provides a special conditional request mechanism that is similar to the If-Match and If-Unmodified-Since header fields but that instructs the recipient to ignore the Range header field if the validator doesn't match, resulting in transfer of the new selected representation instead of a 412 (Precondition Failed) response. If-Range is defined in Section 3.2 of [RFC7233].

"If-Range" 헤더 필드는 If-Match 및 If-Unmodified-Since 헤더 필드와 유사하지만 검증자가 일치하지 않을 경우 수신자에게 Range 헤더 필드를 무시하도록 지시하는 특정 조건부 요청 메커니즘을 제공하여 412(Precondition Failed) 응답 대신 새로운 선택된 표현을 전송한다. If-Range는 [RFC7233]의 Section 3.2에 정의되어 있다.

4. Status Code Definitions

4.1. 304 Not Modified

The 304 (Not Modified) status code indicates that a conditional GET or HEAD request has been received and would have resulted in a 200 (OK) response if it were not for the fact that the condition evaluated to false. In other words, there is no need for the server to transfer a representation of the target resource because the request indicates that the client, which made the request conditional, already has a valid representation; the server is therefore redirecting the client to make use of that stored representation as if it were the payload of a 200 (OK) response.

304 (Not Modified) 상태 코드는 조건부 GET 또는 HEAD 요청이 접수되었음을 나타내며, 상태가 거짓으로 평가되지 않았다면 200 (OK) 응답을 발생시켰을 것이다. 즉, 요청은 요청을 조건부로 한 클라이언트가 이미 유효한 표현을 가지고 있음을 나타내기 때문에 서버가 대상 리소스의 표현을 전송할 필요가 없다. 따라서 서버는 저장된 표현을 200(OK) 응답의 페이로드인 것처럼 사용하도록 클라이언트를 리다이렉트하고 있다.

The server generating a 304 response MUST generate any of the following header fields that would have been sent in a 200 (OK) response to the same request: Cache-Control, Content-Location, Date, ETag, Expires, and Vary.

304 응답을 생성하는 서버는 동일한 요청에 대한 200(OK) 응답으로 전송된 다음 헤더 필드 중 하나를 생성해야 한다.(MUST): Cache-Control, Content-Location, Date, ETag, Expires 및 Vary.

Since the goal of a 304 response is to minimize information transfer when the recipient already has one or more cached representations, a sender SHOULD NOT generate representation metadata other than the above listed fields unless said metadata exists for the purpose of guiding cache updates (e.g., Last-Modified might be useful if the response does not have an ETag field).

304 응답의 목표는 수신자가 이미 하나 이상의 캐시된 표현을 가지고 있을 때 정보 전송을 최소화하는 것이므로, 발신자는 캐시 갱신을 안내하기 위해 언급된 메타데이터가 존재하지 않는 한 위에 나열된 필드 이외의 표현 메타데이터를 생성해서는 안 된다.(SHOULD NOT) (e.g., ETag 필드를 가지고 있지 않은 응답이 있을 경우, Last-Modified가 유용할 수 있다)

Requirements on a cache that receives a 304 response are defined in Section 4.3.4 of [RFC7234]. If the conditional request originated with an outbound client, such as a user agent with its own cache sending a conditional GET to a shared proxy, then the proxy SHOULD forward the 304 response to that client.

304 응답을 수신하는 캐시의 요건은 [RFC7234]의 Section 4.3.4에 정의되어 있다. 조건부 요청이 자신의 캐시를 가진 사용자 에이전트가 공유 프락시에 조건부 GET를 보내는 것과 같은 아웃바운드 클라이언트에서 발생한 경우, 프락시는 304 응답을 해당 클라이언트에 전달해야 한다.

A 304 response cannot contain a message-body; it is always terminated by the first empty line after the header fields.

304 응답은 메시지 본문을 포함할 수 없으며, 헤더 필드 뒤에 있는 첫 번째 빈 줄에 의해 항상 종료된다.

4.2. 412 Precondition Failed

The 412 (Precondition Failed) status code indicates that one or more conditions given in the request header fields evaluated to false when tested on the server. This response code allows the client to place preconditions on the current resource state (its current representations and metadata) and, thus, prevent the request method from being applied if the target resource is in an unexpected state.

412(Precondition Failed) 상태 코드는 서버에서 테스트할 때 요청 헤더 필드에 주어진 하나 이상의 조건이 거짓으로 평가되었음을 나타낸다. 이 응답 코드는 클라이언트가 현재 리소스 상태(현재 표현 및 메타데이터)에 전제조건을 둘 수 있도록 하므로 대상 리소스가 예기치 않은 상태일 경우 요청 메서드가 적용되는것으로 부터 막아야 한다.

5. Evaluation

Except when excluded below, a recipient cache or origin server MUST evaluate received request preconditions after it has successfully performed its normal request checks and just before it would perform the action associated with the request

method. A server MUST ignore all received preconditions if its response to the same request without those conditions would have been a status code other than a 2xx (Successful) or 412 (Precondition Failed). In other words, redirects and failures take precedence over the evaluation of preconditions in conditional requests.

아래에 제외된 경우를 제외하고, 수신자 캐시 또는 원서버는 정상적인 요청 확인을 성공적으로 수행한 후 요청 메서드와 관련된 조치를 수행하기 직전에 수신된 요청 전제조건을 평가해야 한다. 서버가 이러한 조건 없이 동일한 요청에 응답한 것이 2xx (Successful) 또는 412 (Precondition Failed)가 아닌 상태 코드일 경우, 수신된 모든 전제조건을 무시해야 한다.(MUST) 즉, 조건부 요청의 전제조건 평가보다 리다이렉트와 실패가 우선한다.

A server that is not the origin server for the target resource and cannot act as a cache for requests on the target resource MUST NOT evaluate the conditional request header fields defined by this specification, and it MUST forward them if the request is forwarded, since the generating client intends that they be evaluated by a server that can provide a current representation. Likewise, a server MUST ignore the conditional request header fields defined by this specification when received with a request method that does not involve the selection or modification of a selected representation, such as CONNECT, OPTIONS, or TRACE.

대상 리소스의 원서버가 아니므로 대상 리소스의 요청의 캐시 역할을 할 수 없는 서버는 이 명세에 의해 정의된 조건부 요청 헤더 필드를 평가해서는 안 되며,(MUST NOT) 요청이 전달되면 반드시 조건부 요청 헤더 필드들을 전달해야 한다.(MUST) 생성하는 클라이언트는 이러한 요청은 현재 표현을 제공할 수 있는 서버에 의해 평가될 것으로 작정하기 때문이다. 마찬가지로, 서버는 CONNECT, OPTIONS 또는 TRACE와 같이 선택된 표현의 선택이나 수정을 수반하지 않는 요청 메서드로 수신할 때 이 명세에 의해 정의된 조건부 요청 헤더 필드를 무시해야 한다.

Conditional request header fields that are defined by extensions to HTTP might place conditions on all recipients, on the state of the target resource in general, or on a group of resources. For instance, the "If" header field in WebDAV can make a request conditional on various aspects of multiple resources, such as locks, if the recipient understands and implements that field ([RFC4918] Section 10.4).

HTTP에 대한 확장에 의해 정의된 조건부 요청 헤더 필드는 모든 수신자, 일반적으로 대상 리소스의 상태 또는 리소스 그룹에 조건을 배치할 수 있다. 예를 들어 WebDAV의 "If" 헤더 필드는 수신자가 해당 필드를 이해하고 구현하는 경우 잠금과 같은 다중 리소스의 다양한 측면을 조건으로 요청을 할 수 있다([RFC4918] Section 10.4).

Although conditional request header fields are defined as being usable with the HEAD method (to keep HEAD's semantics consistent with those of GET), there is

no point in sending a conditional HEAD because a successful response is around the same size as a 304 (Not Modified) response and more useful than a 412 (Precondition Failed) response.

조건부 요청 헤더 필드는 HEAD 메서드(HEAD의 의미론과 GET의 의미론과 일관성을 유지하기 위해)과 함께 사용할 수 있는 것으로 정의되지만, 성공적인 응답은 304 (Not Modified) 응답과 거의 같고 412 (Precondition Failed) 응답보다 더 유용하기 때문에, 굳이 조건부 HEAD를 보내는 것은 의미가 없다.

6. Precedence

When more than one conditional request header field is present in a request, the order in which the fields are evaluated becomes important. In practice, the fields defined in this document are consistently implemented in a single, logical order, since "lost update" preconditions have more strict requirements than cache validation, a validated cache is more efficient than a partial response, and entity tags are presumed to be more accurate than date validators.

요청에 둘 이상의 조건부 요청 헤더 필드가 있을 때 필드가 평가되는 순서가 중요해진다. 실제로 이 문서에서 정의한 필드는 캐시 유효성 검사보다 "갱신 손실" 전제조건이 더 엄격하고, 검증된 캐시가 부분 응답보다 효율적이며, 엔티티 태그가 날짜 검증자보다 더 정확한 것으로 추정되기 때문에 하나의 논리적 순서로 일관되게 구현된다.

A recipient cache or origin server MUST evaluate the request preconditions defined by this specification in the following order:

수신자 캐시 또는 원서버는 이 명세에 의해 정의된 요청 전제조건을 다음 순서로 평가해야 한다.(MUST)

1. When recipient is the origin server and If-Match is present, evaluate the If-Match precondition:

1. 수신자가 원서버이고 If-Match가 있을 때 If-Match 전제조건을 평가한다.

   if true, continue to step 3

* 참일 경우 3단계로 계속 진행

* if false, respond 412 (Precondition Failed) unless it can be determined that the state-changing request has already succeeded (see Section 3.1)

* 거짓일 경우, 상태 변경 요청이 이미 성공했다고 판단할 수 없는 한 412 (Precondition Failed) 응답한다(Section 3.1 참조).

2. When recipient is the origin server, If-Match is not present, and If-Unmodified-Since is present, evaluate the If-Unmodified-Since precondition:

2. 수신자가 원서버이며, If-Match가 존재하지 않고, If-Unmodified-Since가 있을 경우, If-Unmodified-Since 전제조건:

* if true, continue to step 3

* 참일 경우 3단계로 계속 진행

* if false, respond 412 (Precondition Failed) unless it can be determined that the state-changing request has already succeeded (see Section 3.4)

* 거짓일 경우, 상태 변경 요청이 이미 성공했다고 판단할 수 없는 한 412 (Precondition Failed) 응답한다(Section 3.4 참조).

3. When If-None-Match is present, evaluate the If-None-Match precondition:

3. If-None-Match가 있을 때 If-None-Match 전제조건을 평가한다.

* if true, continue to step 5

* 참일 경우 5단계로 계속 진행

* if false for GET/HEAD, respond 304 (Not Modified)

* GET/HEAD에 대해 거짓일 경우 304 응답(Not Modified)

* if false for other methods, respond 412 (Precondition Failed)

* 기타 메서드에 대해 거짓일 경우 412 응답(Precondition Failed)

4. When the method is GET or HEAD, If-None-Match is not present, and If-Modified-Since is present, evaluate the If-Modified-Since precondition:

4. 메서드가 GET 또는 HEAD일 때, If-None-Match가 존재하지 않고, If-Modified-Since가 있을 때, If-Modified-Since:

* if true, continue to step 5

* 참일 경우 5단계로 계속 진행

* if false, respond 304 (Not Modified)

* 거짓일 경우 304 응답(Not Modified)

5. When the method is GET and both Range and If-Range are present, evaluate the If-Range precondition:

5. 메서드가 GET이고 Range와 If-Range가 모두 있을 때 If-Range 전제조건을 평가한다.

* if the validator matches and the Range specification is applicable to the selected representation, respond 206 (Partial Content) [RFC7233]

* 검증자가 일치하고 Range 명세가 선택된 표현에 적용 가능한 경우, 응답 206(Partial Content) [RFC7233]

6. Otherwise,

그렇지 않으면,

* all conditions are met, so perform the requested action and respond according to its success or failure.

* 모든 조건이 충족되므로 요청된 조치를 수행하고 성공 또는 실패에 따라 대응한다.

Any extension to HTTP/1.1 that defines additional conditional request header fields ought to define its own expectations regarding the order for evaluating such fields in relation to those defined in this document and other conditionals that might be found in practice.

추가 조건부 요청 헤더 필드를 정의하는 HTTP/1.1로의 확장은 이 문서에 정의된 필드 및 실제에서 발견될 수 있는 기타 조건과 관련하여 해당 필드를 평가하는 순서와 관련하여 자체 기대를 정의해야 한다.

7. IANA Considerations

7.1. Status Code Registration

The "Hypertext Transfer Protocol (HTTP) Status Code Registry" located at <http://www.iana.org/assignments/http-status-codes> has been updated with the registrations below:

<http://www.iana.org/assignments/http-status-codes>에 위치한 "Hypertext Transfer Protocol (HTTP) Status Code Registry"는 아래의 등록과 함께 갱신되었다.

Value	Description	Reference
304	Not Modified	Section 4.1
412	Precondition Failed	Section 4.2

7.2. Header Field Registration

HTTP header fields are registered within the "Message Headers" registry maintained at <http://www.iana.org/assignments/message-headers/>.

HTTP 헤더 필드는 <http://www.iana.org/assignments/message-headers/>에서 유지되는 "Message Headers" 레지스트리 내에 등록된다.

This document defines the following HTTP header fields, so their associated registry entries have been updated according to the permanent registrations below (see [BCP90]):

이 문서는 다음 HTTP 헤더 필드를 정의하므로 관련 레지스트리 항목은 아래 영구 등록에 따라 갱신되었다([BCP90] 참조).

Header Field Name	Protocol	Status	Reference
ETag	http	standard	Section 2.3
If-Match	http	standard	Section 3.1
If-Modified-Since	http	standard	Section 3.3
If-None-Match	http	standard	Section 3.2
If-Unmodified-Since	http	standard	Section 3.4
Last-Modified	http	standard	Section 2.2

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

8. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns specific to the HTTP conditional request mechanisms. More general security considerations are addressed in HTTP "Message Syntax and Routing" [RFC7230] and "Semantics and Content" [RFC7231].

이 섹션은 HTTP 조건부 요청 메커니즘에 특정한 알려진 보안 문제를 개발자, 정보 제공자 및 사용자에게 알리기 위한 것이다. 보다 일반적인 보안 고려사항은 HTTP "Message Syntax and Routing"[RFC7230] 및 "Semantics and Content"[RFC7231]에서 다루어진다.

The validators defined by this specification are not intended to ensure the validity of a representation, guard against malicious changes, or detect man-in-the-middle attacks. At best, they enable more efficient cache updates and optimistic concurrent writes when all participants are behaving nicely. At worst, the conditions will fail and the client will receive a response that is no more harmful than an HTTP exchange without conditional requests.

이 명세에 의해 정의된 검증자는 표현의 유효성을 보장하거나, 악의적인 변경을 방지하거나, man-in-the-middle을 탐지하기 위한 것이 아니다. 기껏해야 모든 참가자가 잘 행동하고 있을 때 보다 효율적인 캐시 갱신과 낙관적인 동시 쓰기를 가능하게 한다. 최악의 경우, 조건은 실패하고 클라이언트는 조건부 요청 없이 HTTP 교환보다 더 해롭지 않은 응답을 수신할 것이다.

An entity-tag can be abused in ways that create privacy risks. For example, a site might deliberately construct a semantically invalid entity-tag that is unique to the user or user agent, send it in a cacheable response with a long freshness time, and then read that entity-tag in later conditional requests as a means of re-identifying that user or user agent. Such an identifying tag would become a persistent identifier for as long as the user agent retained the original cache entry. User agents that cache representations ought to ensure that the cache is cleared or replaced whenever the user performs privacy-maintaining actions, such as clearing stored cookies or changing to a private browsing mode.

entity-tag는 프라이버시 위험을 발생시키는 방법으로 남용될 수 있다. 예를 들어 사이트는 사용자 또는 사용자 에이전트에 고유한 의미론적으로 잘못된 entity-tag를 의도적으로 구성하고, 새로 고침 시간이 긴 캐시 가능한 응답으로 보낸 다음, 나중에 해당 사용자 또는 사용자 에이전트를 다시 식별하기 위한 수단으로 조건부 요청에서 entity-tag를 읽을 수 있다. 이러한 식별 태그는 사용자 에이전트가 원래 캐시 항목을 유지하는 한 영구 식별자가 될 것이다. 표현을 캐시하는 사용자 에이전트는 사용자가 저장된 쿠키 삭제 또는 개인 검색 모드로 변경하는 것과 같은 개인 정보 유지 조치를 수행할 때마다 캐시가 삭제되거나 교체되는지 확인해야 한다.

9. Acknowledgments

See Section 10 of [RFC7230].

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, June 2014.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.

10.2. Informative References

[BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext

Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

Appendix A. Changes from RFC 2616

The definition of validator weakness has been expanded and clarified. (Section 2.1)

검증자 '약함'에 대한 정의가 확장되고 명확해졌다. (Section 2.1)

Weak entity-tags are now allowed in all requests except range requests. (Sections 2.1 and 3.2)

이제 범위 요청을 제외한 모든 요청에서 약한 entity-tag가 허용된다. (Section 2.1 및 Section 3.2)

The ETag header field ABNF has been changed to not use quoted-string, thus avoiding escaping issues. (Section 2.3)

ETag 헤더 필드 ABNF가 quoted-string을 사용하지 않도록 변경되어 이스케이핑 문제가 발생하지 않도록 했다. (Section 2.3)

ETag is defined to provide an entity tag for the selected representation, thereby clarifying what it applies to in various situations (such as a PUT response). (Section 2.3)

ETag는 선택된 표현에 대한 엔티티 태그를 제공하도록 정의되어 있어, 다양한 상황(PUT 응답 같은)에 적용되는 내용을 명확히 한다. (Section 2.3)

The precedence for evaluation of conditional requests has been defined. (Section 6)

조건부 요청의 평가 우선순위가 정의되었다.(Section 6)

Appendix B. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII character).

The rules below are defined in [RFC7230]:

```
OWS         = <OWS, see [RFC7230], Section 3.2.3>
obs-text    = <obs-text, see [RFC7230], Section 3.2.6>
```

The rules below are defined in other parts:

```
HTTP-date    = <HTTP-date, see [RFC7231], Section 7.1.1.1>
```

Appendix C. Collected ABNF

In the collected ABNF below, list rules are expanded as per Section 1.2 of [RFC7230].

```
ETag = entity-tag

HTTP-date = <HTTP-date, see [RFC7231], Section 7.1.1.1>

If-Match = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS entity-tag ] ) )
If-Modified-Since = HTTP-date
If-None-Match = "*" / ( *( "," OWS ) entity-tag *( OWS "," [ OWS entity-tag ] ) )
If-Unmodified-Since = HTTP-date

Last-Modified = HTTP-date
```

OWS = <OWS, see [RFC7230], Section 3.2.3>

entity-tag = [weak] opaque-tag
etagc = "!" / %x23-7E ; '#'-'~'
        / obs-text

obs-text = <obs-text, see [RFC7230], Section 3.2.6>
opaque-tag = DQUOTE *etagc DQUOTE

weak = %x57.2F ; W/