

PRACTICAL ENGINEERING REFERENCE

HTTP METHODS & STATUS CODES

A one-stop guide to HTTP semantics, API behavior, testing decisions, debugging, and the codes people confuse most.

COMPLETE REFERENCE

All IANA-registered HTTP methods. All registered HTTP status codes. Explicit treatment of temporary, unused, obsolete, specialized, and unassigned ranges.

Manjunath N P

Edition verified against IANA registries: 23 September 2026

About this reference

Purpose. This book is designed to work in two modes: read it sequentially to learn HTTP, or open it at a specific method/status code during API design, testing, debugging, or incident analysis.

The semantic baseline is the IANA HTTP Method Registry, the IANA HTTP Status Code Registry, RFC 9110, and the extension RFCs referenced by those registries. Examples are intentionally small and implementation-neutral.

How “complete” is defined. Every method currently present in the IANA HTTP Method Registry is included. Every individually registered HTTP status code is included, including codes marked temporary, unused, or obsoleted. Unassigned numeric ranges are summarized separately rather than inventing meanings for them.

Edition-sensitive items. At the edition verification date, status 104 is temporarily registered as *Upload Resumption Supported* and is scheduled to expire on 13 November 2026 unless extended or superseded. The method registry includes QUERY, standardized in RFC 10008.

Engineering note. HTTP semantics describe protocol meaning, not every product-specific API convention. A documented API can impose additional constraints, but such behavior should not silently contradict the underlying HTTP semantics.

How to use the book

Each method and each status code uses a repeatable reading pattern so the page can be scanned quickly under real project pressure.

1. **Identity first** - category, safety/idempotency or status class, commonness, body/retry guidance.
2. **Meaning** - a plain-language statement that stays close to the standard semantics.
3. **When it appears** - practical situations, not abstract definitions.
4. **Wire example** - a compact HTTP request/response pair.
5. **QA lens** - what to verify in functional, negative, concurrency, caching, or resilience tests.
6. **Common misuse** - patterns that frequently create ambiguous APIs or misleading telemetry.
7. **Related entries** - the nearest alternatives to compare.

Fastest lookup path: use the Table of Contents for code number, the alphabetical Index for a concept, and Part VII for “which one should I use?” comparisons.

Contents

About this reference	i
How to use the book	ii
1 HTTP Foundations	1
1.1 The mental model	1
1.2 Anatomy of a request	1
1.3 Anatomy of a response	1
1.4 Safe and idempotent are different	1
1.5 Caching, validators, and conditional requests	1
1.6 Problem details	2
2 HTTP Methods	3
2.1 Master method matrix	3
2.2 ACL	4
2.3 BASELINE-CONTROL	5
2.4 BIND	6
2.5 CHECKIN	7
2.6 CHECKOUT	8
2.7 CONNECT	9
2.8 COPY	10
2.9 DELETE	11
2.10 GET	12
2.11 HEAD	13
2.12 LABEL	14
2.13 LINK	15
2.14 LOCK	16
2.15 MERGE	17
2.16 MKACTIVITY	18
2.17 MKCALENDAR	19
2.18 MKCOL	20
2.19 MKREDIRECTREF	21
2.20 MKWORKSPACE	22
2.21 MOVE	23
2.22 OPTIONS	24
2.23 ORDERPATCH	25
2.24 PATCH	26
2.25 POST	27
2.26 PRI	28
2.27 PROPFIND	29
2.28 PROPPATCH	30
2.29 PUT	31
2.30 QUERY	32
2.31 REBIND	33
2.32 REPORT	34
2.33 SEARCH	35
2.34 TRACE	36
2.35 UNBIND	37
2.36 UNCHECKOUT	38
2.37 UNLINK	39
2.38 UNLOCK	40
2.39 UPDATE	41
2.40 UPDATEREDIRECTREF	42
2.41 VERSION-CONTROL	43
2.42 ASTERISK (*)	44

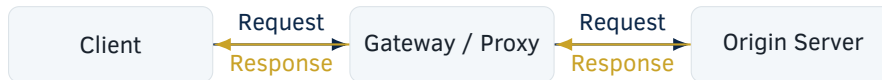
3	1xx - Informational Responses	45
3.1	100 Continue	46
3.2	101 Switching Protocols	47
3.3	102 Processing	48
3.4	103 Early Hints	49
3.5	104 Upload Resumption Supported	50
4	2xx - Successful Responses	51
4.1	200 OK	52
4.2	201 Created	53
4.3	202 Accepted	54
4.4	203 Non-Authoritative Information	55
4.5	204 No Content	56
4.6	205 Reset Content	57
4.7	206 Partial Content	58
4.8	207 Multi-Status	59
4.9	208 Already Reported	60
4.10	226 IM Used	61
5	3xx - Redirection Responses	62
5.1	300 Multiple Choices	63
5.2	301 Moved Permanently	64
5.3	302 Found	65
5.4	303 See Other	66
5.5	304 Not Modified	67
5.6	305 Use Proxy	68
5.7	306 Unused	69
5.8	307 Temporary Redirect	70
5.9	308 Permanent Redirect	71
6	4xx - Client Error Responses	72
6.1	400 Bad Request	73
6.2	401 Unauthorized	74
6.3	402 Payment Required	75
6.4	403 Forbidden	76
6.5	404 Not Found	77
6.6	405 Method Not Allowed	78
6.7	406 Not Acceptable	79
6.8	407 Proxy Authentication Required	80
6.9	408 Request Timeout	81
6.10	409 Conflict	82
6.11	410 Gone	83
6.12	411 Length Required	84
6.13	412 Precondition Failed	85
6.14	413 Content Too Large	86
6.15	414 URI Too Long	87
6.16	415 Unsupported Media Type	88
6.17	416 Range Not Satisfiable	89
6.18	417 Expectation Failed	90
6.19	418 Unused	91
6.20	421 Misdirected Request	92
6.21	422 Unprocessable Content	93
6.22	423 Locked	94
6.23	424 Failed Dependency	95
6.24	425 Too Early	96
6.25	426 Upgrade Required	97
6.26	428 Precondition Required	98
6.27	429 Too Many Requests	99
6.28	431 Request Header Fields Too Large	100
6.29	451 Unavailable For Legal Reasons	101
7	5xx - Server Error Responses	102

7.1	500 Internal Server Error	103
7.2	501 Not Implemented	104
7.3	502 Bad Gateway	105
7.4	503 Service Unavailable	106
7.5	504 Gateway Timeout	107
7.6	505 HTTP Version Not Supported	108
7.7	506 Variant Also Negotiates	109
7.8	507 Insufficient Storage	110
7.9	508 Loop Detected	111
7.10	510 Not Extended	112
7.11	511 Network Authentication Required	113
8	Unused, Obsolete, Temporary, and Unassigned Numbers	114
8.1	Why these labels matter	114
8.2	Current unassigned ranges	114
9	The Confusing Codes - Decision Guide	115
9.1	200 vs 201 vs 202 vs 204	116
9.2	201 vs 202	117
9.3	204 vs 205	118
9.4	301 vs 308	119
9.5	302 vs 303 vs 307	120
9.6	304 vs 204	121
9.7	400 vs 422	122
9.8	401 vs 403	123
9.9	403 vs 404	124
9.10	404 vs 410	125
9.11	405 vs 501	126
9.12	408 vs 504	127
9.13	409 vs 412	128
9.14	409 vs 422	129
9.15	413 vs 414 vs 431	130
9.16	406 vs 415	131
9.17	428 vs 412	132
9.18	429 vs 503	133
9.19	500 vs 502 vs 503 vs 504	134
9.20	501 vs 505	135
10	The Confusing Methods - Design Guide	136
10.1	POST vs PUT	136
10.2	PUT vs PATCH	137
10.3	GET vs HEAD	138
10.4	OPTIONS vs HEAD	139
10.5	Safe vs idempotent vs retry-safe	140
11	HTTP for API Testing and Debugging	141
11.1	A status-code assertion is not enough	141
11.2	Negative test families	141
11.3	Debug 5xx by drawing the hop chain	142
11.4	Retry decision checklist	142
11.5	Contract-review smell: “always return 200”	143
12	Which Status Code Fits This Situation?	144
13	Master Status Code Table	145
14	Glossary	147
	Official References	148

HTTP Foundations

The mental model

HTTP is an application-layer request-response protocol. A client sends a request message to a target; a server sends a response message. Intermediaries such as proxies, gateways, caches, and CDNs can participate between them. The method says what kind of semantics the client is requesting. The status code says how the recipient reports the outcome.



Anatomy of a request

```
PATCH /profiles/42 HTTP/1.1
Host: api.example.com
Authorization: Bearer <token>
Content-Type: application/merge-patch+json
If-Match: "v7"

{"city":"Bengaluru"}
```

The request line combines a **method**, a **request target**, and a protocol version. Header fields carry metadata and conditions. Some requests carry content whose media type defines how the bytes should be interpreted.

Anatomy of a response

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "v8"

{"id":42,"city":"Bengaluru"}
```

The status code is protocol metadata. The response content can carry a representation, a problem description, or nothing at all depending on the status and method semantics.

Safe and idempotent are different

Concept	Meaning	Practical testing implication
Safe	The client does not request a state change as the intended semantics.	Repeating a safe read should not intentionally modify server state. Logging/metrics can still change.
Idempotent	Multiple identical requests have the same intended effect as one request.	Network retry can be safer for idempotent operations, but side effects and responses can still vary.

Caching, validators, and conditional requests

HTTP becomes much easier to reason about when ETag, Last-Modified, If-Match, If-None-Match, Range, and Retry-After are treated as part of the protocol rather than optional trivia. Several “confusing” status codes only make sense in this larger context: 206 and 416 are range mechanics; 304 and 412 are conditional-request mechanics; 428 is a concurrency-policy signal; 429 and 503 often rely on Retry-After.

Problem details

For API errors, a structured problem representation such as `application/problem+json` can make 4xx/5xx responses far more actionable. Status codes should remain accurate at the HTTP layer; the body can then add domain-specific detail such as a validation field, trace identifier, remediation link, or retry guidance.

HTTP Methods

Master method matrix

The registry contains core HTTP methods plus methods defined by WebDAV, CalDAV, versioning, bindings, search/-query extensions, and protocol machinery. “Safe” and “Idempotent” values below follow the IANA registry.

Method	Safe	Idempotent	Area
ACL	No	Yes	WebDAV ACL
BASELINE-CONTROL	No	Yes	WebDAV versioning
BIND	No	Yes	WebDAV bindings
CHECKIN	No	Yes	WebDAV versioning
CHECKOUT	No	Yes	WebDAV versioning
CONNECT	No	No	Core HTTP
COPY	No	Yes	WebDAV
DELETE	No	Yes	Core HTTP
GET	Yes	Yes	Core HTTP
HEAD	Yes	Yes	Core HTTP
LABEL	No	Yes	WebDAV versioning
LINK	No	Yes	Historic extension
LOCK	No	No	WebDAV
MERGE	No	Yes	WebDAV versioning
MKACTIVITY	No	Yes	WebDAV versioning
MKCALENDAR	No	Yes	CalDAV
MKCOL	No	Yes	WebDAV
MKREDIRECTREF	No	Yes	WebDAV redirect references
MKWORKSPACE	No	Yes	WebDAV versioning
MOVE	No	Yes	WebDAV
OPTIONS	Yes	Yes	Core HTTP
ORDERPATCH	No	Yes	WebDAV ordered collections
PATCH	No	No	Core extension
POST	No	No	Core HTTP
PRI	Yes	Yes	HTTP/2 protocol token
PROPFIND	Yes	Yes	WebDAV
PROPPATCH	No	Yes	WebDAV
PUT	No	Yes	Core HTTP
QUERY	Yes	Yes	HTTP query method
REBIND	No	Yes	WebDAV bindings
REPORT	Yes	Yes	WebDAV versioning / CalDAV
SEARCH	Yes	Yes	WebDAV search
TRACE	Yes	Yes	Core HTTP
UNBIND	No	Yes	WebDAV bindings
UNCHECKOUT	No	Yes	WebDAV versioning
UNLINK	No	Yes	Historic extension
UNLOCK	No	Yes	WebDAV
UPDATE	No	Yes	WebDAV versioning
UPDATEREDIRECTREF	No	Yes	WebDAV redirect references
VERSION-CONTROL	No	Yes	WebDAV versioning
*	No	No	Reserved method token

ACL

ACL

WebDAV ACL

Modifies the access control list of a WebDAV resource.

Safe	No	Idempotent	Yes
Reference	RFC 3744, Section 8.1		

What it means

Modifies the access control list of a WebDAV resource.

Where it fits

- Modifies the access control list of a WebDAV resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
ACL /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

BASELINE-CONTROL

BASELINE-CONTROL

WebDAV versioning

Places a collection under baseline control or adjusts baseline-control state.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 12.6		

What it means

Places a collection under baseline control or adjusts baseline-control state.

Where it fits

- Places a collection under baseline control or adjusts baseline-control state.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
BASELINE-CONTROL /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

BIND

BIND

WebDAV bindings

Creates a binding from a name to an existing resource so the resource can be reached through another URI.

Safe	No	Idempotent	Yes
Reference	RFC 5842, Section 4		

What it means

Creates a binding from a name to an existing resource so the resource can be reached through another URI.

Where it fits

- Creates a binding from a name to an existing resource so the resource can be reached through another URI.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
BIND /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

CHECKIN

CHECKIN

WebDAV versioning

Checks in a checked-out version-controlled resource, producing/updating version history state.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 9.4		

What it means

Checks in a checked-out version-controlled resource, producing/updating version history state.

Where it fits

- Checks in a checked-out version-controlled resource, producing/updating version history state.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
CHECKIN /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

CHECKOUT

CHECKOUT

WebDAV versioning

Checks out a version-controlled resource so it can be modified under versioning semantics.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 8.8		

What it means

Checks out a version-controlled resource so it can be modified under versioning semantics.

Where it fits

- Checks out a version-controlled resource so it can be modified under versioning semantics.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
CHECKOUT /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

CONNECT

CONNECT

Core HTTP

Requests that the recipient establish a tunnel to the destination identified by the request target.

Safe	No	Idempotent	No
Reference	RFC 9110, Section 9.3.6		

What it means

Requests that the recipient establish a tunnel to the destination identified by the request target.

Where it fits

- HTTP proxies commonly use CONNECT for tunnel establishment, including TLS tunneling.

Request content

Protocol-specific; not ordinary resource representation semantics.

Typical response behavior

2xx indicates successful tunnel establishment; proxy/auth errors may occur.

Wire example

```
CONNECT example.com:443 HTTP/1.1
Host: example.com:443
```

```
HTTP/1.1 200 Connection Established
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. CONNECT changes the connection into a tunnel after success; ordinary HTTP message handling no longer describes tunneled bytes.

COPY

COPY

WebDAV

Copies a resource to a destination URI under WebDAV semantics.

Safe	No	Idempotent	Yes
Reference	RFC 4918, Section 9.8		

What it means

Copies a resource to a destination URI under WebDAV semantics.

Where it fits

- Copies a resource to a destination URI under WebDAV semantics.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
COPY /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

DELETE

DELETE

Core HTTP

Requests removal of the association between the target URI and its current functionality/state.

Safe	No	Idempotent	Yes
Reference	RFC 9110, Section 9.3.5		

What it means

Requests removal of the association between the target URI and its current functionality/state.

Where it fits

- Delete or deactivate a resource according to API semantics.

Request content

A request body has no generally defined DELETE semantics in HTTP unless a specific API defines one carefully.

Typical response behavior

Common results: 200, 202, 204, 404, 409, 412.

Wire example

```
DELETE /orders/981 HTTP/1.1
```

```
HTTP/1.1 204 No Content
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Idempotency means repeating the intended delete has the same intended effect; later responses can still differ, such as 204 then 404.

GET

GET

Core HTTP

Retrieves a current representation of the target resource.

Safe	Yes	Idempotent	Yes
Reference	RFC 9110, Section 9.3.1		

What it means

Retrieves a current representation of the target resource.

Where it fits

- Read a resource or collection.
- Use conditional requests and ranges when caching or partial transfer matters.

Request content

A GET request content body has no generally defined semantics in HTTP; avoid relying on one unless a specific protocol defines it.

Typical response behavior

Common results: 200, 206, 304, 404, 406.

Wire example

```
GET /books/42 HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{"id":42,"title":"HTTP Field Guide"}
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use GET for state-changing actions. Sensitive query data may appear in logs/history.

HEAD

HEAD

Core HTTP

Requests the same response metadata as GET but without transferring the response content.

Safe	Yes	Idempotent	Yes
Reference	RFC 9110, Section 9.3.2		

What it means

Requests the same response metadata as GET but without transferring the response content.

Where it fits

- Check metadata, validators, content length, or existence without downloading the representation.

Request content

Normally no request content semantics; response has no message content.

Typical response behavior

Common results mirror GET metadata statuses such as 200, 304, 404.

Wire example

```
HEAD /manual.pdf HTTP/1.1
```

```
HTTP/1.1 200 OK
Content-Type: application/pdf
Content-Length: 2480012
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Server metadata should correspond closely to what GET would return.

LABEL

LABEL

WebDAV versioning

Adds, sets, or removes a label associated with a version in a version history.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 8.2		

What it means

Adds, sets, or removes a label associated with a version in a version history.

Where it fits

- Adds, sets, or removes a label associated with a version in a version history.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
LABEL /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

LINK

LINK

Historic extension

Historic registered method for establishing links between resources; retained in the registry for compatibility/reference.

Safe	No	Idempotent	Yes
Reference	RFC 2068, Section 19.6.1.2		

What it means

Historic registered method for establishing links between resources; retained in the registry for compatibility/reference.

Where it fits

- Historic registered method for establishing links between resources; retained in the registry for compatibility/reference.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
LINK /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

LOCK

LOCK

WebDAV

Creates or refreshes a lock on a resource.

Safe	No	Idempotent	No
Reference	RFC 4918, Section 9.10		

What it means

Creates or refreshes a lock on a resource.

Where it fits

- Creates or refreshes a lock on a resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
LOCK /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MERGE

MERGE

WebDAV versioning

Merges a version-controlled configuration into a version-controlled resource or workspace.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 11.2		

What it means

Merges a version-controlled configuration into a version-controlled resource or workspace.

Where it fits

- Merges a version-controlled configuration into a version-controlled resource or workspace.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MERGE /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MKACTIVITY

MKACTIVITY

WebDAV versioning

Creates an activity resource used to group related versioning changes.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 13.5		

What it means

Creates an activity resource used to group related versioning changes.

Where it fits

- Creates an activity resource used to group related versioning changes.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MKACTIVITY /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MKCALENDAR

MKCALENDAR

Creates a new calendar collection.

CalDAV

Safe	No	Idempotent	Yes
Reference	RFC 4791, Section 5.3.1; RFC 8144, Section 2.3		

What it means

Creates a new calendar collection.

Where it fits

- Creates a new calendar collection.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MKCALENDAR /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MKCOL

MKCOL

WebDAV

Creates a new collection resource; extensions can allow creation with additional properties/content.

Safe	No	Idempotent	Yes
Reference	RFC 4918, Section 9.3; RFC 5689; RFC 8144		

What it means

Creates a new collection resource; extensions can allow creation with additional properties/content.

Where it fits

- Creates a new collection resource; extensions can allow creation with additional properties/content.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MKCOL /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MKREDIRECTREF

MKREDIRECTREF

WebDAV redirect references

Creates a redirect reference resource.

Safe	No	Idempotent	Yes
Reference	RFC 4437, Section 6		

What it means

Creates a redirect reference resource.

Where it fits

- Creates a redirect reference resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MKREDIRECTREF /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MKWORKSPACE

MKWORKSPACE

WebDAV versioning

Creates a workspace resource for version-controlled editing.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 6.3		

What it means

Creates a workspace resource for version-controlled editing.

Where it fits

- Creates a workspace resource for version-controlled editing.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MKWORKSPACE /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

MOVE

MOVE

WebDAV

Moves a resource to a destination URI.

Safe	No	Idempotent	Yes
Reference	RFC 4918, Section 9.9		

What it means

Moves a resource to a destination URI.

Where it fits

- Moves a resource to a destination URI.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
MOVE /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

OPTIONS

OPTIONS

Core HTTP

Describes communication options and capabilities for the target resource or server.

Safe	Yes	Idempotent	Yes
Reference	RFC 9110, Section 9.3.7		

What it means

Describes communication options and capabilities for the target resource or server.

Where it fits

- CORS preflight requests.
- Discover allowed methods/capabilities.

Request content

May carry protocol-defined content, but common use is header-driven.

Typical response behavior

Often 200 or 204 with Allow/CORS capability headers.

Wire example

```
OPTIONS /orders HTTP/1.1
Origin: https://app.example
Access-Control-Request-Method: POST
```

```
HTTP/1.1 204 No Content
Allow: GET, POST, OPTIONS
Access-Control-Allow-Methods: GET, POST
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. OPTIONS does not guarantee a complete machine-readable description of an API.

ORDERPATCH

ORDERPATCH

WebDAV ordered collections

Changes the ordering of members in an ordered collection.

Safe	No	Idempotent	Yes
Reference	RFC 3648, Section 7		

What it means

Changes the ordering of members in an ordered collection.

Where it fits

- Changes the ordering of members in an ordered collection.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
ORDERPATCH /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

PATCH

PATCH

Core extension

Applies a set of changes to a resource rather than replacing the complete representation.

Safe	No	Idempotent	No
Reference	RFC 5789, Section 2		

What it means

Applies a set of changes to a resource rather than replacing the complete representation.

Where it fits

- JSON Patch, JSON Merge Patch, or an application-specific patch format.

Request content

Carries patch instructions whose media type defines semantics.

Typical response behavior

Common results: 200, 204, 409, 412, 415, 422.

Wire example

```
PATCH /profiles/42 HTTP/1.1
Content-Type: application/merge-patch+json

{"city":"Mysuru"}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{"name":"Mira","city":"Mysuru"}
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. PATCH is not inherently idempotent; some patch documents may be designed to be idempotent. Always define and validate the patch media type.

POST

POST

Core HTTP

Submits data to the target resource for resource-specific processing.

Safe	No	Idempotent	No
Reference	RFC 9110, Section 9.3.3		

What it means

Submits data to the target resource for resource-specific processing.

Where it fits

- Create subordinate resources.
- Trigger commands, workflows, searches with complex submitted criteria, or processing actions when POST semantics fit.

Request content

Commonly carries a representation.

Typical response behavior

Common results: 200, 201, 202, 204, 303, plus relevant 4xx/5xx.

Wire example

```
POST /orders HTTP/1.1
Content-Type: application/json

{"sku": "BK-42", "qty": 1}
```

```
HTTP/1.1 201 Created
Location: /orders/981
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Repeating POST can duplicate effects unless the application adds an idempotency mechanism. Do not assume POST always means “create.”

PRI

PRI

HTTP/2 protocol token

The HTTP/2 connection preface contains PRI; it is registered so HTTP/1.x recipients can recognize it. It is not an application method for normal API design.

Safe	Yes	Idempotent	Yes
Reference	RFC 9113, Section 3.4		

What it means

The HTTP/2 connection preface contains PRI; it is registered so HTTP/1.x recipients can recognize it. It is not an application method for normal API design.

Where it fits

- The HTTP/2 connection preface contains PRI; it is registered so HTTP/1.x recipients can recognize it. It is not an application method for normal API design.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
PRI /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. This is not a normal application API method. Treat it as protocol/registry machinery, not a custom business verb.

PROPFIND

PROPFIND

WebDAV

Retrieves properties and, depending on Depth, information about collection members.

Safe	Yes	Idempotent	Yes
Reference	RFC 4918, Section 9.1; RFC 8144, Section 2.1		

What it means

Retrieves properties and, depending on Depth, information about collection members.

Where it fits

- Retrieves properties and, depending on Depth, information about collection members.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
PROPFIND /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

PROPPATCH

PROPPATCH

WebDAV

Sets and/or removes properties defined for a WebDAV resource.

Safe	No	Idempotent	Yes
Reference	RFC 4918, Section 9.2; RFC 8144, Section 2.2		

What it means

Sets and/or removes properties defined for a WebDAV resource.

Where it fits

- Sets and/or removes properties defined for a WebDAV resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
PROPPATCH /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

PUT

PUT

Core HTTP

Creates or completely replaces the state of the target resource with the supplied representation, according to resource semantics.

Safe	No	Idempotent	Yes
Reference	RFC 9110, Section 9.3.4		

What it means

Creates or completely replaces the state of the target resource with the supplied representation, according to resource semantics.

Where it fits

- Client knows the target URI.
- Full replacement/update designs where repeating the same representation should have the same intended effect.

Request content

Usually carries the desired resource representation.

Typical response behavior

Common results: 200, 201, 204, 409, 412, 415, 422.

Wire example

```
PUT /profiles/42 HTTP/1.1
Content-Type: application/json

{"name": "Mira", "city": "Bengaluru"}
```

```
HTTP/1.1 204 No Content
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not automatically model partial field updates as PUT. Idempotent does not mean every repeated response is identical.

QUERY

QUERY

HTTP query method

Requests safe, idempotent processing of query content against a target resource, providing standardized semantics for queries that do not fit conveniently in a URI.

Safe	Yes	Idempotent	Yes
Reference	RFC 10008, Section 2		

What it means

Requests safe, idempotent processing of query content against a target resource, providing standardized semantics for queries that do not fit conveniently in a URI.

Where it fits

- Requests safe, idempotent processing of query content against a target resource, providing standardized semantics for queries that do not fit conveniently in a URI.
- Useful when a safe query needs a request representation rather than a very long or sensitive query string.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
QUERY /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

REBIND

REBIND

WebDAV bindings

Moves an existing binding to a new binding name while preserving the bound resource.

Safe	No	Idempotent	Yes
Reference	RFC 5842, Section 6		

What it means

Moves an existing binding to a new binding name while preserving the bound resource.

Where it fits

- Moves an existing binding to a new binding name while preserving the bound resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
REBIND /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

REPORT

REPORT

WebDAV versioning / CalDAV

Requests information about resource state using a report whose semantics are defined by the applicable extension.

Safe	Yes	Idempotent	Yes
Reference	RFC 3253, Section 3.6; RFC 8144, Section 2.1		

What it means

Requests information about resource state using a report whose semantics are defined by the applicable extension.

Where it fits

- Requests information about resource state using a report whose semantics are defined by the applicable extension.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
REPORT /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

SEARCH

SEARCH

WebDAV search

Performs a server-side search using a search grammar defined by the WebDAV SEARCH specification.

Safe	Yes	Idempotent	Yes
Reference	RFC 5323, Section 2		

What it means

Performs a server-side search using a search grammar defined by the WebDAV SEARCH specification.

Where it fits

- Performs a server-side search using a search grammar defined by the WebDAV SEARCH specification.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
SEARCH /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

TRACE

TRACE

Core HTTP

Performs a diagnostic message loop-back so the requester can see what was received along the request chain.

Safe	Yes	Idempotent	Yes
Reference	RFC 9110, Section 9.3.8		

What it means

Performs a diagnostic message loop-back so the requester can see what was received along the request chain.

Where it fits

- Protocol diagnostics in controlled environments.

Request content

TRACE request content is constrained by HTTP semantics.

Typical response behavior

Successful TRACE returns the received request message as diagnostic content.

Wire example

```
TRACE /diagnostic HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: message/http

TRACE /diagnostic HTTP/1.1 ...
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Frequently disabled for security/privacy reasons. Never assume production servers expose TRACE.

UNBIND

UNBIND

WebDAV bindings

Removes a binding without necessarily deleting the underlying bound resource.

Safe	No	Idempotent	Yes
Reference	RFC 5842, Section 5		

What it means

Removes a binding without necessarily deleting the underlying bound resource.

Where it fits

- Removes a binding without necessarily deleting the underlying bound resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UNBIND /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

UNCHECKOUT

UNCHECKOUT

WebDAV versioning

Cancels a checkout and restores the version-controlled resource to its checked-in state.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 4.5		

What it means

Cancels a checkout and restores the version-controlled resource to its checked-in state.

Where it fits

- Cancels a checkout and restores the version-controlled resource to its checked-in state.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UNCHECKOUT /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

UNLINK

UNLINK

Historic extension

Historic counterpart to LINK for removing links; retained in the method registry.

Safe	No	Idempotent	Yes
Reference	RFC 2068, Section 19.6.1.3		

What it means

Historic counterpart to LINK for removing links; retained in the method registry.

Where it fits

- Historic counterpart to LINK for removing links; retained in the method registry.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UNLINK /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

UNLOCK

UNLOCK

WebDAV

Removes an active lock identified by a lock token.

Safe	No	Idempotent	Yes
Reference	RFC 4918, Section 9.11		

What it means

Removes an active lock identified by a lock token.

Where it fits

- Removes an active lock identified by a lock token.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UNLOCK /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

UPDATE

UPDATE

WebDAV versioning

Updates a version-controlled resource to a specified version/configuration.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 7.1		

What it means

Updates a version-controlled resource to a specified version/configuration.

Where it fits

- Updates a version-controlled resource to a specified version/configuration.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UPDATE /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

UPDATEREDIRECTREF

UPDATEREDIRECTREF

WebDAV redirect references

Updates the target of an existing redirect reference resource.

Safe	No	Idempotent	Yes
Reference	RFC 4437, Section 7		

What it means

Updates the target of an existing redirect reference resource.

Where it fits

- Updates the target of an existing redirect reference resource.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
UPDATEREDIRECTREF /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

VERSION-CONTROL

VERSION-CONTROL

WebDAV versioning

Places a resource under version control or performs version-control-related initialization.

Safe	No	Idempotent	Yes
Reference	RFC 3253, Section 3.5		

What it means

Places a resource under version control or performs version-control-related initialization.

Where it fits

- Places a resource under version control or performs version-control-related initialization.
- Relevant only when the corresponding WebDAV/CalDAV extension is implemented by both client and server.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
VERSION-CONTROL /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. Do not use this method in a general REST API merely because the verb name sounds convenient; interoperability depends on the registered semantics.

ASTERISK (*)

ASTERISK (*)

Reserved method token

Reserved registry entry associated with HTTP method extensibility; not a normal method to use in application requests.

Safe	No	Idempotent	No
Reference	RFC 9110, Section 18.2		

What it means

Reserved registry entry associated with HTTP method extensibility; not a normal method to use in application requests.

Where it fits

- Reserved registry entry associated with HTTP method extensibility; not a normal method to use in application requests.

Request content

Defined by the method specification; many extension methods carry XML or another extension-defined request representation.

Typical response behavior

Status codes and response formats are defined by the relevant extension; 2xx, 207, and extension-specific 4xx statuses are common in WebDAV families.

Wire shape

```
* /resource HTTP/1.1
Host: dav.example.com
[extension-defined headers/content]
```

QA / implementation lens

- Verify the server advertises or documents support before depending on the method.
- Assert status, required headers, representation semantics, authorization, and repeated-request behavior.
- For extension methods, test the extension-specific preconditions and multi-resource outcomes defined by the referenced RFC.

Common trap. This is not a normal application API method. Treat it as protocol/registry machinery, not a custom business verb.

1xx - Informational Responses

Family idea. The request was received and processing/negotiation continues.

Code	Meaning	Use profile
100	Continue	Specialized
101	Switching Protocols	Specialized
102	Processing	Rare / WebDAV
103	Early Hints	Specialized
104	Upload Resumption Supported	Temporary / experimental

100 Continue

100 Continue

The server has accepted the request headers and tells the client it may send the request body.

Profile	Specialized	Retry lens	Continue the same request
Response body	No final representation	Typical methods	POST, PUT
Reference	RFC 9110, Section 15.2.1		

When it appears

- Used with the Expect: 100-continue request header before sending a potentially large body.
- Allows a server to reject a request from headers alone without forcing the client to upload the full payload.

Wire example

```
POST /uploads HTTP/1.1
Host: api.example.com
Expect: 100-continue
Content-Length: 52428800
```

```
HTTP/1.1 100 Continue
```

```
[client now sends the body]
```

QA lens

- Verify the client treats the response as interim and continues until a final response arrives.
- Confirm proxies/clients do not mis-handle the interim response as final application data.

Common misuse. Treating 100 as the final result. A final status response still follows. Using it as a generic progress indicator for normal APIs.

Compare with: 101 Switching Protocols | 417 Expectation Failed

101 Switching Protocols

101 Switching Protocols

The server agrees to switch the connection to a protocol requested through Upgrade.

Profile	Specialized	Retry lens	Not a retry signal
Response body	Normally no application representation	Typical methods	GET / upgrade handshake
Reference	RFC 9110, Section 15.2.2		

When it appears

- Classic HTTP/1.1 protocol upgrade flows, such as a WebSocket handshake.
- The response identifies the selected protocol in Upgrade.

Wire example

```
GET /chat HTTP/1.1
Connection: Upgrade
Upgrade: websocket
```

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: websocket
```

QA lens

- Verify the client treats the response as interim and continues until a final response arrives.
- Confirm proxies/clients do not mis-handle the interim response as final application data.

Common misuse. Returning 101 for an HTTP/2 connection established by other negotiation mechanisms. Assuming the request is complete in the usual request-response sense after the switch.

Compare with: 426 Upgrade Required

102 Processing

102 Processing

An interim WebDAV response indicating that the server has accepted the request but has not completed processing.

Profile	Rare / WebDAV	Retry lens	Wait for the final response
Response body	No final representation	Typical methods	WebDAV methods
Reference	RFC 2518		

When it appears

- Originally defined for long-running WebDAV operations where silence could otherwise look like a timeout.
- It is informational; a final response is still required.

Wire example

```
PROPFIND /archive HTTP/1.1
Depth: infinity
```

```
HTTP/1.1 102 Processing
```

QA lens

- Verify the client treats the response as interim and continues until a final response arrives.
- Confirm proxies/clients do not mis-handle the interim response as final application data.

Common misuse. Using 102 as an asynchronous-job API response. 202 Accepted is the normal semantic for accepted-but-not-completed work.

Compare with: 202 Accepted

103 Early Hints

103 Early Hints

An interim response that lets a server send selected response headers before the final response is ready.

Profile	Specialized	Retry lens	Continue waiting
Response body	No final representation	Typical methods	Usually GET
Reference	RFC 8297		

When it appears

- Often used to expose Link preload/preconnect hints while the origin is still preparing the final response.
- The final response can still be 2xx, 3xx, 4xx, or 5xx.

Wire example

```
GET /home HTTP/1.1
Host: www.example.com
```

```
HTTP/1.1 103 Early Hints
Link: </app.css>; rel=preload; as=style

HTTP/1.1 200 OK
```

QA lens

- Verify the client treats the response as interim and continues until a final response arrives.
- Confirm proxies/clients do not mis-handle the interim response as final application data.

Common misuse. Treating early hints as authoritative final metadata. Putting security-sensitive final-response assumptions into the interim response.

Compare with: 100 Continue | 200 OK

104 Upload Resumption Supported

104 Upload Resumption Supported

A temporary status registered for the resumable-upload work in progress; it signals support in the draft protocol.

Profile	Temporary / experimental	Retry lens	Follow the draft protocol
Response body	Protocol-specific	Typical methods	Upload flows
Reference	draft-ietf-httpbis-resumable-upload-05 (temporary IANA registration)		

When it appears

- Appears in the IANA registry as a temporary registration tied to the resumable-upload draft.
- The registration currently expires on November 13, 2026 unless extended or replaced by a final specification.

Wire example

```
POST /uploads HTTP/1.1
Upload-Complete: ?0
Content-Length: 0
```

```
HTTP/1.1 104 Upload Resumption Supported
[protocol-specific fields]
```

QA lens

- Verify the client treats the response as interim and continues until a final response arrives.
- Confirm proxies/clients do not mis-handle the interim response as final application data.

Common misuse. Treating 104 as a stable, universally deployed status code. Implementing it without checking the current draft and IANA registry.

Compare with: 100 Continue | 202 Accepted

Edition note: Edition note: temporary registration status verified against IANA on 2026-09-23.

2xx - Successful Responses

Family idea. The request was successfully received, understood, and accepted.

Code	Meaning	Use profile
200	OK	Very common
201	Created	Very common
202	Accepted	Common
203	Non-Authoritative Information	Rare
204	No Content	Very common
205	Reset Content	Rare
206	Partial Content	Common in media/- downloads
207	Multi-Status	Rare / WebDAV
208	Already Reported	Rare / WebDAV
226	IM Used	Very rare

200 OK

200 OK

The request succeeded; the response semantics depend on the method.

Profile	Very common	Retry lens	Not because of the status alone
Response body	Usually yes; method-dependent	Typical methods	GET, POST, PUT, PATCH, others
Reference	RFC 9110, Section 15.3.1		

When it appears

- GET normally returns the requested representation.
- POST can return the result of an action when no more specific success code is required.
- PUT/PATCH may return the updated representation.

Wire example

```
GET /books/42 HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{"id":42,"title":"HTTP Field Guide"}
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Returning 200 with an error object for failures that should be represented by a 4xx/5xx status. Using 200 for creation when 201 communicates resource creation more precisely.

Compare with: 201 Created | 204 No Content | 206 Partial Content

201 Created

201 Created

The request succeeded and created one or more resources.

Profile	Very common	Retry lens	Do not blindly retry creation
Response body	Usually useful	Typical methods	POST, PUT
Reference	RFC 9110, Section 15.3.2		

When it appears

- POST creates a new subordinate resource.
- PUT creates a resource at the target URI when it did not previously exist.
- A Location header commonly identifies the primary created resource.

Wire example

```
POST /orders HTTP/1.1
Content-Type: application/json

{"sku": "BK-42", "qty": 1}
```

```
HTTP/1.1 201 Created
Location: /orders/981
Content-Type: application/json

{"id": 981, "status": "created"}
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Returning 201 when no resource was created. Omitting a useful Location header when the created resource has a URI.

Compare with: 200 OK | 202 Accepted | 204 No Content

202 Accepted

202 Accepted

The request has been accepted for processing, but processing is not complete.

Profile	Common	Retry lens	Depends on operation design
Response body	Often status metadata or job link	Typical methods	POST, PUT, DELETE
Reference	RFC 9110, Section 15.3.3		

When it appears

- Long-running exports, bulk processing, or queued work.
- The server is not promising eventual success; acceptance and completion are separate states.

Wire example

```
POST /reports HTTP/1.1
Content-Type: application/json

{"range": "2026-Q3"}
```

```
HTTP/1.1 202 Accepted
Location: /jobs/abc123

{"jobId": "abc123", "state": "queued"}
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Using 202 when work has already completed. Returning 202 without giving the client a practical way to observe progress or result when one exists.

Compare with: 201 Created | 200 OK | 303 See Other

203 Non-Authoritative Information

203 Non-Authoritative Information

The request succeeded, but the enclosed metadata differs from what the origin server would have supplied.

Profile	Rare	Retry lens	No special retry
Response body	Usually yes	Typical methods	Usually GET
Reference	RFC 9110, Section 15.3.4		

When it appears

- An intermediary transforms metadata or representation information received from the origin.
- Useful when a transforming proxy wants to make that distinction visible.

Wire example

```
GET /profile HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 203 Non-Authoritative Information
Content-Type: application/json
```

```
{"name": "Asha"}
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Using 203 simply because data came from a cache. Confusing 203 with stale content or an authentication warning.

Compare with: 200 OK | 214 Transformation Applied warning semantics in older ecosystems

204 No Content

204 No Content

The request succeeded and there is no response content to send.

Profile	Very common	Retry lens	No special retry
Response body	No message content	Typical methods	PUT, PATCH, DELETE, POST
Reference	RFC 9110, Section 15.3.5		

When it appears

- Updates or deletes that succeed without returning a representation.
- Useful when the client should stay on the current view and no response body is needed.

Wire example

```
DELETE /cart/items/7 HTTP/1.1
```

```
HTTP/1.1 204 No Content
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Sending a response body and expecting clients to process it. Using 204 for a newly created resource where 201 conveys more information.

Compare with: 200 OK | 201 Created | 205 Reset Content | 304 Not Modified

205 Reset Content

205 Reset Content

The request succeeded and the client should reset the document view that caused the request.

Profile	Rare	Retry lens	No special retry
Response body	No content	Typical methods	Typically form-style POST
Reference	RFC 9110, Section 15.3.6		

When it appears

- Designed for user-agent workflows where the server wants the input view reset after successful processing.
- More specific than 204 because it carries a user-interface instruction.

Wire example

```
POST /feedback HTTP/1.1
Content-Type: application/x-www-form-urlencoded

rating=5
```

```
HTTP/1.1 205 Reset Content
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Treating 205 as interchangeable with 204. Returning content that the client is expected to render.

Compare with: 204 No Content

206 Partial Content

206 Partial Content

The server successfully fulfilled a range request and is returning only the selected part of the representation.

Profile	Common in media/downloads	Retry lens	Range-specific
Response body	Yes, partial	Typical methods	GET
Reference	RFC 9110, Section 15.3.7		

When it appears

- Byte-range downloads, streaming, resumable downloads, and segmented media transfer.
- The response uses Content-Range and related range metadata.

Wire example

```
GET /video.mp4 HTTP/1.1
Range: bytes=0-999999
```

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-999999/8000000
Content-Length: 1000000
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Returning 206 without a valid range request/selection. Confusing 206 with an application-level paginated JSON response.

Compare with: 200 OK | 416 Range Not Satisfiable

207 Multi-Status

207 Multi-Status

A WebDAV response carrying multiple independent status results for multiple resources or operations.

Profile	Rare / WebDAV	Retry lens	Per embedded result
Response body	Yes, structured multi-status	Typical methods	WebDAV methods
Reference	RFC 4918		

When it appears

- One request affects or inspects multiple resources and each needs its own outcome.
- The body commonly uses the WebDAV multistatus representation.

Wire example

```
PROPFIND /collection/ HTTP/1.1
Depth: 1
```

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml

<multistatus>...</multistatus>
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Using 207 as a generic batch-API status without defining compatible semantics. Assuming the top-level 207 means every embedded operation succeeded.

Compare with: 200 OK | 424 Failed Dependency

208 Already Reported

208 Already Reported

A WebDAV status used inside a DAV multistatus response to avoid repeatedly enumerating members of the same binding.

Profile	Rare / WebDAV	Retry lens	Not a retry signal
Response body	Usually embedded in 207	Typical methods	WebDAV PROPFIND
Reference	RFC 5842		

When it appears

- Prevents duplicate reporting when bindings make the same collection appear through multiple paths.
- Normally encountered only in WebDAV depth traversal.

Wire example

```
PROPFIND /collection/ HTTP/1.1
Depth: infinity
```

```
HTTP/1.1 207 Multi-Status
... <status>HTTP/1.1 208 Already Reported</status> ...
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Returning 208 as a normal top-level success for ordinary REST APIs.

Compare with: 207 Multi-Status | 508 Loop Detected

226 IM Used

226 IM Used

The server fulfilled a GET using instance manipulations and returned the resulting representation.

Profile	Very rare	Retry lens	No special retry
Response body	Yes	Typical methods	GET
Reference	RFC 3229		

When it appears

- Defined by delta encoding / instance manipulation mechanisms.
- Only meaningful when the relevant negotiation and instance-manipulation semantics are in use.

Wire example

```
GET /resource HTTP/1.1
A-IM: diffe
```

```
HTTP/1.1 226 IM Used
IM: diffe

[delta-encoded representation]
```

QA lens

- Assert the response actually matches the success semantics, not merely a 2xx number.
- Verify required headers and representation/body rules.
- Test the nearest neighboring success code so the contract communicates the right outcome.

Common misuse. Using 226 as a general “processed successfully” code.

Compare with: 200 OK | 206 Partial Content

3xx - Redirection Responses

Family idea. Further action, alternate target, or cache reuse is involved.

Code	Meaning	Use profile
300	Multiple Choices	Uncommon
301	Moved Permanently	Very common
302	Found	Very common
303	See Other	Common
304	Not Modified	Very common in caching
305	Use Proxy	Deprecated / avoid
306	Unused	Unused
307	Temporary Redirect	Common
308	Permanent Redirect	Common

300 Multiple Choices

300 Multiple Choices

Multiple representations or targets are available and the client or user can select among them.

Profile	Uncommon	Retry lens	Follow selected alternative
Response body	Usually helpful	Typical methods	GET, HEAD
Reference	RFC 9110, Section 15.4.1		

When it appears

- Content negotiation cannot choose a single preferred representation.
- The response can provide alternatives and may include a preferred choice.

Wire example

```
GET /manual HTTP/1.1
Accept-Language: *
```

```
HTTP/1.1 300 Multiple Choices
Content-Type: application/json

{"choices":["/manual/en","/manual/fr"]}
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Using 300 when the server already knows exactly where the client should go.

Compare with: 301 Moved Permanently | 302 Found

301 Moved Permanently

301 Moved Permanently

The target resource has a new permanent URI.

Profile	Very common	Retry lens	Follow redirect according to client policy
Response body	Often minimal	Typical methods	GET/HEAD; can occur for others
Reference	RFC 9110, Section 15.4.2		

When it appears

- Canonical URL migrations and permanent route changes.
- A Location header normally identifies the new URI.

Wire example

```
GET /old-docs HTTP/1.1
```

```
HTTP/1.1 301 Moved Permanently
Location: /docs
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Assuming every client preserves a non-GET method across 301. For method-preserving permanent redirects, 308 is clearer. Using permanent redirects for temporary routing decisions.

Compare with: 302 Found | 307 Temporary Redirect | 308 Permanent Redirect

302 Found

302 Found

The target resource temporarily resides at another URI.

Profile	Very common	Retry lens	Follow redirect according to client policy
Response body	Often minimal	Typical methods	Any, historically common with GET
Reference	RFC 9110, Section 15.4.3		

When it appears

- Temporary navigation where the original URI remains the canonical target.
- User-agent history makes method rewriting behavior less strict than 307.

Wire example

```
GET /signin HTTP/1.1
```

```
HTTP/1.1 302 Found
Location: /login
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Choosing 302 when preserving the original method is required. Treating a temporary redirect as a permanent canonical migration.

Compare with: 301 Moved Permanently | 303 See Other | 307 Temporary Redirect

303 See Other

303 See Other

The server directs the client to retrieve another resource, normally with GET or HEAD.

Profile	Common	Retry lens	Follow Location
Response body	Usually minimal	Typical methods	Often after POST
Reference	RFC 9110, Section 15.4.4		

When it appears

- Post/Redirect/Get flows after an action.
- Directing a client to a separate status or result resource.

Wire example

```
POST /payments HTTP/1.1
amount=25
```

```
HTTP/1.1 303 See Other
Location: /payments/receipt/123
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Using 303 when the original method must be repeated at the new URI.

Compare with: 302 Found | 307 Temporary Redirect

304 Not Modified

304 Not Modified

A conditional GET or HEAD can reuse its cached representation because it has not changed.

Profile	Very common in caching	Retry lens	Reuse cached representation
Response body	No representation body	Typical methods	GET, HEAD
Reference	RFC 9110, Section 15.4.5		

When it appears

- The client sends validators such as If-None-Match or If-Modified-Since.
- The server confirms the cached response remains valid.

Wire example

```
GET /app.js HTTP/1.1
If-None-Match: "v42"
```

```
HTTP/1.1 304 Not Modified
ETag: "v42"
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Treating 304 as a normal 3xx redirect. Comparing it to 204 as if both simply mean “empty body”; 304 is cache-validation specific.

Compare with: 200 OK | 204 No Content | 412 Precondition Failed

305 Use Proxy

305 Use Proxy

A deprecated redirect status that indicated the requested resource had to be accessed through a proxy.

Profile	Deprecated / avoid	Retry lens	Do not design new APIs around it
Response body	Minimal	Typical methods	Historical
Reference	RFC 9110, Section 15.4.6		

When it appears

- Kept in the registry for protocol history and compatibility awareness.
- Modern HTTP applications should not introduce new behavior that depends on 305.

Wire example

```
GET /resource HTTP/1.1
```

```
HTTP/1.1 305 Use Proxy
Location: http://proxy.example.net:8080
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Using 305 for modern proxy discovery or corporate proxy configuration.

Compare with: 407 Proxy Authentication Required

306 Unused

306 Unused

A reserved historical status code that is no longer used.

Profile	Unused	Retry lens	Not applicable
Response body	Not applicable	Typical methods	None
Reference	RFC 9110, Section 15.4.7		

When it appears

- Included so readers can recognize that the number is intentionally not assigned normal semantics.

Wire example

```
[no valid modern request pattern]
```

```
HTTP/1.1 306 [Unused]
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Inventing application semantics for 306.

Compare with: 305 Use Proxy | 307 Temporary Redirect

307 Temporary Redirect

307 Temporary Redirect

The target is temporarily available at another URI and the client must preserve the request method when redirecting.

Profile	Common	Retry lens	Follow Location, preserving method
Response body	Usually minimal	Typical methods	Any method
Reference	RFC 9110, Section 15.4.8		

When it appears

- Temporary relocation when repeating the same method and body at the new URI is required.
- Useful for POST/PUT/PATCH redirects where method rewriting would be incorrect.

Wire example

```
POST /v1/orders HTTP/1.1
```

```
{"sku": "A1"}
```

```
HTTP/1.1 307 Temporary Redirect
Location: /v2/orders
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Using 302 when strict method preservation is required.

Compare with: 302 Found | 308 Permanent Redirect

308 Permanent Redirect

308 Permanent Redirect

The target has permanently moved and the client must preserve the request method when redirecting.

Profile	Common	Retry lens	Follow Location, preserving method
Response body	Usually minimal	Typical methods	Any method
Reference	RFC 9110, Section 15.4.9		

When it appears

- Permanent endpoint migration without changing POST/PUT/PATCH into GET.
- Communicates permanence plus method preservation.

Wire example

```
PUT /v1/profile HTTP/1.1
```

```
{"name": "Nina"}
```

```
HTTP/1.1 308 Permanent Redirect  
Location: /v2/profile
```

QA lens

- Verify Location or validators where the status requires/uses them.
- Test method preservation and cache behavior, not just the status number.
- Confirm clients do not create redirect loops or silently change unsafe methods.

Common misuse. Using 301 when strict method preservation is important.

Compare with: 301 Moved Permanently | 307 Temporary Redirect

4xx - Client Error Responses

Family idea. The request cannot be fulfilled as presented; the precise reason matters.

Code	Meaning	Use profile
400	Bad Request	Very common
401	Unauthorized	Very common
402	Payment Required	Reserved / application-specific
403	Forbidden	Very common
404	Not Found	Very common
405	Method Not Allowed	Common
406	Not Acceptable	Uncommon
407	Proxy Authentication Required	Specialized
408	Request Timeout	Common at infrastructure edge
409	Conflict	Very common
410	Gone	Uncommon but useful
411	Length Required	Rare
412	Precondition Failed	Common in concurrency/caching
413	Content Too Large	Common
414	URI Too Long	Uncommon
415	Unsupported Media Type	Very common
416	Range Not Satisfiable	Common in range-capable resources
417	Expectation Failed	Rare
418	Unused	Unused
421	Misdirected Request	Specialized
422	Unprocessable Content	Very common in modern APIs
423	Locked	Rare / WebDAV
424	Failed Dependency	Rare / WebDAV
425	Too Early	Specialized
426	Upgrade Required	Specialized
428	Precondition Required	Common in concurrency-aware APIs
429	Too Many Requests	Very common
431	Request Header Fields Too Large	Uncommon
451	Unavailable For Legal Reasons	Specialized

400 Bad Request

400 Bad Request

The server cannot or will not process the request because the request itself is invalid at a general HTTP/application boundary.

Profile	Very common	Retry lens	Fix request before retry
Response body	Problem details recommended	Typical methods	Any
Reference	RFC 9110, Section 15.5.1		

When it appears

- Malformed syntax, invalid framing, invalid request-target, or general request validation that is not better represented by a more specific status.
- A practical API may use 400 for malformed JSON or structurally invalid input.

Wire example

```
POST /users HTTP/1.1
Content-Type: application/json

{"name":
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{"title":"Invalid JSON"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 400 as the universal code for every client-side problem even when 401, 403, 404, 409, 415, or 422 is more precise.

Compare with: 401 Unauthorized | 409 Conflict | 415 Unsupported Media Type | 422 Unprocessable Content

401 Unauthorized

401 Unauthorized

Authentication credentials are missing, invalid, or otherwise insufficient to authenticate the request.

Profile	Very common	Retry lens	Retry after obtaining/refreshing credentials
Response body	Problem details may help	Typical methods	Any protected endpoint
Reference	RFC 9110, Section 15.5.2		

When it appears

- The request needs authentication or the supplied credentials are not accepted.
- The response normally includes WWW-Authenticate challenges when applicable.

Wire example

```
GET /account HTTP/1.1
Authorization: Bearer expired-token
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 401 for a known authenticated user who lacks permission; that is normally 403. Interpreting the reason phrase “Unauthorized” as authorization failure instead of authentication semantics.

Compare with: 403 Forbidden | 407 Proxy Authentication Required

402 Payment Required

402 Payment Required

A reserved status intended for future use; HTTP itself does not define a standard payment workflow around it.

Profile	Reserved / application-specific	Retry lens	Application-defined
Response body	Application-defined	Typical methods	Any
Reference	RFC 9110, Section 15.5.3		

When it appears

- The code is registered but general HTTP semantics do not standardize how payment should be requested, completed, or verified.
- Some APIs use it conventionally, but interoperability depends on application documentation.

Wire example

```
POST /premium-export HTTP/1.1
```

```
HTTP/1.1 402 Payment Required
Content-Type: application/problem+json
{"title":"Plan upgrade required"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Assuming all clients understand a universal payment protocol because 402 was returned.

Compare with: 403 Forbidden | 429 Too Many Requests

403 Forbidden

403 Forbidden

The server understood the request and the requester identity, if any, but refuses to fulfill it.

Profile	Very common	Retry lens	Retry only after permissions/policy change
Response body	Problem details can help	Typical methods	Any
Reference	RFC 9110, Section 15.5.4		

When it appears

- Authenticated principal lacks required permission or policy blocks access.
- The server may intentionally avoid explaining sensitive authorization details.

Wire example

```
DELETE /admin/users/7 HTTP/1.1
Authorization: Bearer valid-user-token
```

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json

{"title":"Insufficient permission"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 401 when authentication succeeded but authorization failed. Exposing sensitive policy details that help an attacker enumerate protected resources.

Compare with: 401 Unauthorized | 404 Not Found

404 Not Found

404 Not Found

The server cannot find a current representation/target matching the requested resource, or chooses not to disclose that one exists.

Profile	Very common	Retry lens	Retry only if resource may appear later or URI was wrong
Response body	Problem details useful	Typical methods	Any
Reference	RFC 9110, Section 15.5.5		

When it appears

- Unknown resource identifier, removed route, or hidden resource where disclosure is intentionally avoided.
- Does not necessarily mean the resource never existed.

Wire example

```
GET /books/999999 HTTP/1.1
```

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json

{"title":"Book not found"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Returning 404 for a known permanently removed resource when 410 provides useful permanence semantics. Leaking existence through inconsistent 403/404 behavior when policy aims to hide it.

Compare with: 403 Forbidden | 410 Gone

405 Method Not Allowed

405 Method Not Allowed

The target resource exists, but the request method is not supported for that resource.

Profile	Common	Retry lens	Use an allowed method
Response body	Problem details helpful	Typical methods	Any
Reference	RFC 9110, Section 15.5.6		

When it appears

- Calling DELETE on a read-only resource or POST on a route that only permits GET.
- The response must include an Allow header listing currently supported methods for the target.

Wire example

```
DELETE /reports/annual HTTP/1.1
```

```
HTTP/1.1 405 Method Not Allowed
Allow: GET, HEAD
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 404 simply because the route table has no handler for that method. Forgetting the Allow header.

Compare with: 404 Not Found | 501 Not Implemented

406 Not Acceptable

406 Not Acceptable

The server cannot produce a representation acceptable under the request content-negotiation preferences.

Profile	Uncommon	Retry lens	Change Accept/preferences
Response body	Problem details or alternatives	Typical methods	Typically GET/HEAD
Reference	RFC 9110, Section 15.5.7		

When it appears

- The client requests only media types/languages/encodings the server cannot supply and the server chooses not to send a default representation.

Wire example

```
GET /report HTTP/1.1
Accept: application/x-proprietary-format
```

```
HTTP/1.1 406 Not Acceptable
Content-Type: application/problem+json
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing 406 with 415. 406 concerns the representation the client wants to receive; 415 concerns the representation the client sent.

Compare with: 415 Unsupported Media Type

407 Proxy Authentication Required

407 Proxy Authentication Required

A proxy requires authentication before forwarding the request.

Profile	Specialized	Retry lens	Retry with proxy credentials
Response body	Usually challenge metadata	Typical methods	Through an authenticating proxy
Reference	RFC 9110, Section 15.5.8		

When it appears

- Enterprise or controlled proxy requires credentials.
- Analogous to 401, but authentication is with the intermediary, not the origin server.

Wire example

```
GET http://example.com/ HTTP/1.1
```

```
HTTP/1.1 407 Proxy Authentication Required
Proxy-Authenticate: Basic realm="CorpProxy"
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 407 for origin-server login failures.

Compare with: 401 Unauthorized

408 Request Timeout

408 Request Timeout

The server did not receive a complete request within the time it was prepared to wait.

Profile	Common at infrastructure edge	Retry lens	Retry may be reasonable if idempotent/safe
Response body	Usually minimal	Typical methods	Any
Reference	RFC 9110, Section 15.5.9		

When it appears

- Client sends headers/body too slowly or connection stalls before request completion.
- The failure happens while receiving the request, not while waiting for an upstream response.

Wire example

```
POST /upload HTTP/1.1
[request transmission stalls]
```

```
HTTP/1.1 408 Request Timeout
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing 408 with 504. 504 is generated by a gateway/proxy waiting on an upstream server.

Compare with: 504 Gateway Timeout

409 Conflict

409 Conflict

The request cannot be completed because it conflicts with the current state of the target resource.

Profile	Very common	Retry lens	Resolve state conflict before retry
Response body	Problem details strongly recommended	Typical methods	POST, PUT, PATCH, DELETE
Reference	RFC 9110, Section 15.5.10		

When it appears

- Version/state conflict, duplicate state transition, or resource relationship conflict.
- The response should ideally help the client recognize the source of conflict.

Wire example

```
PUT /documents/42 HTTP/1.1
If-Match: "v7"

{"title":"Updated"}
```

```
HTTP/1.1 409 Conflict
Content-Type: application/problem+json

{"title":"Edit conflict","currentVersion":"v9"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 409 for malformed input. Using 422 for a resource-state conflict when the representation itself is valid.

Compare with: 412 Precondition Failed | 422 Unprocessable Content

410 Gone

410 Gone

The target resource is no longer available and this condition is expected to be permanent.

Profile	Uncommon but useful	Retry lens	Do not keep retrying unchanged URI
Response body	Problem details optional	Typical methods	Any
Reference	RFC 9110, Section 15.5.11		

When it appears

- Intentional permanent removal where communicating permanence is useful.
- Search engines/crawlers and clients can distinguish it from potentially temporary 404.

Wire example

```
GET /campaigns/retired-2024 HTTP/1.1
```

```
HTTP/1.1 410 Gone
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 410 when the server does not know whether the absence is permanent.

Compare with: 404 Not Found

411 Length Required

411 Length Required

The server refuses the request because it requires a defined Content-Length for the request content.

Profile	Rare	Retry lens	Retry with required framing
Response body	Minimal	Typical methods	Requests with content
Reference	RFC 9110, Section 15.5.12		

When it appears

- A server or gateway requires a known content length for a particular request.

Wire example

```
POST /legacy-upload HTTP/1.1
Transfer-Encoding: chunked

...
```

```
HTTP/1.1 411 Length Required
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 411 as a generic “body missing” validation error.

Compare with: 400 Bad Request | 413 Content Too Large

412 Precondition Failed

412 Precondition Failed

One or more request preconditions evaluated to false on the server.

Profile	Common in concurrency/caching	Retry lens	Refresh state, then decide
Response body	Problem details useful	Typical methods	PUT, PATCH, DELETE, GET
Reference	RFC 9110, Section 15.5.13		

When it appears

- If-Match fails because the entity tag is stale.
- If-Unmodified-Since or other conditional request requirements are not met.

Wire example

```
PATCH /profile HTTP/1.1
If-Match: "v4"

{"city": "Pune"}
```

```
HTTP/1.1 412 Precondition Failed
ETag: "v5"
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Replacing precise conditional-request semantics with generic 409 when an HTTP precondition header specifically failed.

Compare with: 304 Not Modified | 409 Conflict | 428 Precondition Required

413 Content Too Large

413 Content Too Large

The request content is larger than the server is willing or able to process.

Profile	Common	Retry lens	Retry only with smaller content or changed limit
Response body	Problem details useful	Typical methods	POST, PUT, PATCH
Reference	RFC 9110, Section 15.5.14		

When it appears

- Upload exceeds configured body-size limit.
- The server may close the connection depending on circumstances.

Wire example

```
POST /attachments HTTP/1.1
Content-Length: 524288000
```

```
HTTP/1.1 413 Content Too Large
Content-Type: application/problem+json

{"title":"Maximum upload size is 25 MB"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Calling this 414; 414 is about the URI, not the body.

Compare with: 414 URI Too Long | 431 Request Header Fields Too Large

414 URI Too Long

414 URI Too Long

The request-target/URI is longer than the server is willing to interpret.

Profile	Uncommon	Retry lens	Shorten URI; often move data out of query string
Response body	Problem details optional	Typical methods	Usually GET
Reference	RFC 9110, Section 15.5.15		

When it appears

- Extremely long query string or accidental request construction.
- Can reveal that a design is sending large payload-like data through the URL.

Wire example

```
GET /search?q=[very-long-value...] HTTP/1.1
```

```
HTTP/1.1 414 URI Too Long
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing URI length with request body size.

Compare with: 413 Content Too Large | 431 Request Header Fields Too Large

415 Unsupported Media Type

415 Unsupported Media Type

The server refuses the request because the request content format or content coding is not supported.

Profile	Very common	Retry lens	Retry with supported Content-Type/encoding
Response body	Problem details useful	Typical methods	POST, PUT, PATCH
Reference	RFC 9110, Section 15.5.16		

When it appears

- Sending XML to an endpoint that only consumes JSON.
- Unsupported content coding or representation format.

Wire example

```
POST /users HTTP/1.1
Content-Type: application/xml

<user/>
```

```
HTTP/1.1 415 Unsupported Media Type
Accept-Post: application/json
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing 415 with 406. 415 is about what the client sent; 406 is about what the client asks to receive.

Compare with: 400 Bad Request | 406 Not Acceptable | 422 Unprocessable Content

416 Range Not Satisfiable

416 Range Not Satisfiable

The requested range cannot be satisfied for the selected representation.

Profile	Common in range-capable resources	Retry lens	Correct range or refetch metadata
Response body	Minimal; include range metadata where required	Typical methods	GET
Reference	RFC 9110, Section 15.5.17		

When it appears

- Request starts beyond the current representation length.
- Range syntax/selection cannot be fulfilled.

Wire example

```
GET /video.mp4 HTTP/1.1
Range: bytes=9000000-9999999
```

```
HTTP/1.1 416 Range Not Satisfiable
Content-Range: bytes */8000000
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 416 for application pagination outside the HTTP Range mechanism.

Compare with: 206 Partial Content

417 Expectation Failed

417 Expectation Failed

The server cannot meet an expectation given in the Expect request header.

Profile	Rare	Retry lens	Retry without/with supported expectation
Response body	Minimal	Typical methods	Any with Expect
Reference	RFC 9110, Section 15.5.18		

When it appears

- Typically associated with Expect: 100-continue or extension expectations that cannot be met.

Wire example

```
POST /upload HTTP/1.1
Expect: custom-feature
```

```
HTTP/1.1 417 Expectation Failed
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 417 for ordinary business-rule expectations in request JSON.

Compare with: 100 Continue

418 Unused

418 Unused

A code reserved as unused in the current HTTP registry; it should not be assigned normal application semantics.

Profile	Unused	Retry lens	Not applicable
Response body	Not applicable	Typical methods	None
Reference	RFC 9110, Section 15.5.19		

When it appears

- The number is widely known from an earlier April Fools specification, but the current IANA HTTP status registry marks 418 as unused.

Wire example

```
[no standards-based modern request pattern]
```

```
HTTP/1.1 418 [Unused]
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Treating the popular “I’m a teapot” behavior as a current general HTTP standard status.

Compare with: 400 Bad Request

421 Misdirected Request

421 Misdirected Request

The request reached a server that cannot produce a response for the requested scheme/authority combination.

Profile	Specialized	Retry lens	Retry on another connection/server may help
Response body	Problem details optional	Typical methods	Any
Reference	RFC 9110, Section 15.5.20		

When it appears

- Connection reuse or routing sends a request to an origin that is not authoritative for that target.
- Especially relevant to connection coalescing and intermediaries.

Wire example

```
GET / HTTP/1.1
Host: api.other.example
```

```
HTTP/1.1 421 Misdirected Request
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 421 for an ordinary application route miss.

Compare with: 400 Bad Request | 502 Bad Gateway

422 Unprocessable Content

422 Unprocessable Content

The server understands the content type and syntax, but cannot process the contained instructions.

Profile	Very common in modern APIs	Retry lens	Fix semantic/validation problems before retry
Response body	Detailed field errors useful	Typical methods	POST, PUT, PATCH
Reference	RFC 9110, Section 15.5.21		

When it appears

- Well-formed JSON fails domain validation or contains semantically invalid instructions.
- The server understood the representation; the issue is what the content asks it to do.

Wire example

```
POST /users HTTP/1.1
Content-Type: application/json

{"email":"not-an-email"}
```

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/problem+json

{"title":"Validation failed","errors":{"email":"Invalid format"}}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 422 for malformed JSON that the server cannot parse. Using 422 for resource-state conflicts that are better modeled as 409.

Compare with: 400 Bad Request | 409 Conflict | 415 Unsupported Media Type

423 Locked

423 Locked

The source or destination resource is locked and the operation cannot proceed.

Profile	Rare / WebDAV	Retry lens	Wait, unlock, or supply valid lock token
Response body	Problem details / DAV response	Typical methods	WebDAV methods
Reference	RFC 4918		

When it appears

- Attempting to modify a WebDAV resource protected by an active lock.

Wire example

```
PUT /docs/spec.docx HTTP/1.1
```

```
[new content]
```

```
HTTP/1.1 423 Locked
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 423 as a generic database-row-lock error without defining compatible semantics.

Compare with: 409 Conflict | 424 Failed Dependency

424 Failed Dependency

424 Failed Dependency

An operation failed because a related operation on which it depended failed.

Profile	Rare / WebDAV	Retry lens	Resolve the dependency first
Response body	Often within DAV multi-status	Typical methods	WebDAV methods
Reference	RFC 4918		

When it appears

- Part of a multi-operation WebDAV request cannot proceed because an earlier dependent action failed.

Wire example

```
PROPPATCH /resource HTTP/1.1  
[multiple dependent property changes]
```

```
HTTP/1.1 424 Failed Dependency
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 424 for arbitrary microservice dependency failures; 502/503/504 may be more appropriate at HTTP gateway boundaries.

Compare with: 207 Multi-Status | 502 Bad Gateway

425 Too Early

425 Too Early

The server is unwilling to risk processing a request that might be replayed, particularly with TLS early data.

Profile	Specialized	Retry lens	Retry after handshake without early data
Response body	Minimal	Typical methods	Unsafe/non-idempotent requests especially
Reference	RFC 8470		

When it appears

- Protects operations from replay risks associated with early data.
- The client can retry after the connection is fully established.

Wire example

```
POST /payments HTTP/1.1
Early-Data: 1

{"amount":100}
```

```
HTTP/1.1 425 Too Early
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 425 to mean “the business event happened too soon.”

Compare with: 409 Conflict | 429 Too Many Requests

426 Upgrade Required

426 Upgrade Required

The server refuses the current protocol but is willing to service the request after the client upgrades to a listed protocol.

Profile	Specialized	Retry lens	Retry using supported protocol
Response body	Upgrade metadata	Typical methods	Any
Reference	RFC 9110, Section 15.5.22		

When it appears

- Protocol-version/upgrade policy requires a different protocol.
- The response should indicate acceptable protocol(s) using Upgrade.

Wire example

```
GET /realtime HTTP/1.1
```

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/3.0
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 426 for application version upgrades such as “install version 5 of the mobile app” without explicit HTTP upgrade semantics.

Compare with: 101 Switching Protocols | 505 HTTP Version Not Supported

428 Precondition Required

428 Precondition Required

The origin server requires the request to be conditional to prevent lost updates.

Profile	Common in concurrency-aware APIs	Retry lens	Retry with required precondition
Response body	Problem details useful	Typical methods	PUT, PATCH, DELETE
Reference	RFC 6585		

When it appears

- Server requires If-Match or another precondition before accepting a state-changing request.
- Useful for enforcing optimistic concurrency.

Wire example

```
PUT /docs/42 HTTP/1.1
```

```
{"title":"New"}
```

```
HTTP/1.1 428 Precondition Required
Content-Type: application/problem+json
```

```
{"title":"Send If-Match"}
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing 428 with 412. 428 means the required condition was missing; 412 means a supplied precondition evaluated false.

Compare with: 412 Precondition Failed

429 Too Many Requests

429 Too Many Requests

The client has sent too many requests in a given period or exceeded an applicable rate policy.

Profile	Very common	Retry lens	Retry according to Retry-After/policy
Response body	Problem details helpful	Typical methods	Any
Reference	RFC 6585		

When it appears

- Per-user, per-token, per-IP, or per-resource throttling.
- Retry-After can communicate when another attempt is appropriate.

Wire example

```
GET /search?q=http HTTP/1.1
Authorization: Bearer token
```

```
HTTP/1.1 429 Too Many Requests
Retry-After: 30
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 503 for a client-specific quota/rate limit when 429 communicates the condition precisely. Retrying aggressively without backoff.

Compare with: 503 Service Unavailable

431 Request Header Fields Too Large

431 Request Header Fields Too Large

The server refuses the request because header fields are too large individually or collectively.

Profile	Uncommon	Retry lens	Reduce headers/cookies before retry
Response body	Problem details optional	Typical methods	Any
Reference	RFC 6585		

When it appears

- Oversized cookies, authorization metadata, tracing headers, or combined header set.

Wire example

```
GET / HTTP/1.1
Cookie: [very-large-cookie-set]
```

```
HTTP/1.1 431 Request Header Fields Too Large
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Confusing header size with URI length or request body size.

Compare with: 413 Content Too Large | 414 URI Too Long

451 Unavailable For Legal Reasons

451 Unavailable For Legal Reasons

Access is denied because of a legal demand or legal restriction.

Profile	Specialized	Retry lens	Retry only if legal/access context changes
Response body	Explanation/link may be provided when appropriate	Typical methods	Any
Reference	RFC 7725		

When it appears

- Content blocked due to a court order, statute, regulatory requirement, or similar legal constraint.
- The response can identify the blocking party or legal demand when safe and lawful to do so.

Wire example

```
GET /restricted-article HTTP/1.1
```

```
HTTP/1.1 451 Unavailable For Legal Reasons
Link: <https://example.com/legal/notice/123>; rel="blocked-by"
```

QA lens

- Trigger the condition deliberately with a negative or boundary test.
- Validate that the response does not leak sensitive internals while still giving actionable client guidance.
- Confirm a corrected request can move to the intended success path.

Common misuse. Using 451 for ordinary authorization, policy, moderation, or licensing rules that are not legal restrictions.

Compare with: 403 Forbidden

5xx - Server Error Responses

Family idea. The server or an intermediary failed to fulfill an apparently valid request.

Code	Meaning	Use profile
500	Internal Server Error	Very common
501	Not Implemented	Uncommon
502	Bad Gateway	Very common in distributed systems
503	Service Unavailable	Very common
504	Gateway Timeout	Very common in distributed systems
505	HTTP Version Not Supported	Rare
506	Variant Also Negotiates	Very rare
507	Insufficient Storage	Rare / WebDAV
508	Loop Detected	Rare / WebDAV
510	Not Extended	Obsoleted
511	Network Authentication Required	Specialized

500 Internal Server Error

500 Internal Server Error

The server encountered an unexpected condition that prevented it from fulfilling the request.

Profile	Very common	Retry lens	Retry may help, but avoid blind repetition of non-idempotent operations
Response body	Safe problem details; hide internals	Typical methods	Any
Reference	RFC 9110, Section 15.6.1		

When it appears

- Unhandled exception, invariant failure, or unexpected server-side fault where no more specific 5xx status applies.

Wire example

```
GET /account/summary HTTP/1.1
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/problem+json

{"title":"Unexpected server error","traceId":"abc-123"}
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Leaking stack traces, secrets, SQL, or infrastructure internals. Returning 500 for client input errors that should be 4xx.

Compare with: 502 Bad Gateway | 503 Service Unavailable

501 Not Implemented

501 Not Implemented

The server does not support the functionality required to fulfill the request, commonly an unrecognized/unsupported method.

Profile	Uncommon	Retry lens	Retry only after server capability changes
Response body	Problem details optional	Typical methods	Any
Reference	RFC 9110, Section 15.6.2		

When it appears

- Server implementation does not implement the request method or a required capability.
- Different from 405: 405 means the method is known but not allowed for this target.

Wire example

```
F00 /resource HTTP/1.1
```

```
HTTP/1.1 501 Not Implemented
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 501 for a feature temporarily disabled. 503 is often more appropriate for temporary unavailability.

Compare with: 405 Method Not Allowed | 505 HTTP Version Not Supported

502 Bad Gateway

502 Bad Gateway

A gateway or proxy received an invalid response from an upstream server while trying to fulfill the request.

Profile	Very common in distributed systems	Retry lens	Retry with backoff may help
Response body	Minimal/problem details	Typical methods	Any through intermediary
Reference	RFC 9110, Section 15.6.3		

When it appears

- Reverse proxy cannot interpret or accept the upstream response.
- Upstream closes connection incorrectly or returns malformed protocol data.

Wire example

```
GET /catalog HTTP/1.1
```

```
HTTP/1.1 502 Bad Gateway
Content-Type: application/problem+json

{"title":"Upstream response failure"}
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Calling every downstream application error 502. The key is that the intermediary failed while acting as a gateway/proxy.

Compare with: 500 Internal Server Error | 503 Service Unavailable | 504 Gateway Timeout

503 Service Unavailable

503 Service Unavailable

The server is currently unable to handle the request because of temporary overload or maintenance.

Profile	Very common	Retry lens	Retry with backoff; honor Retry-After
Response body	Problem details optional	Typical methods	Any
Reference	RFC 9110, Section 15.6.4		

When it appears

- Maintenance window, capacity protection, temporary dependency/circuit-breaker condition surfaced as service unavailability.
- Retry-After can indicate expected recovery time.

Wire example

```
GET /checkout HTTP/1.1
```

```
HTTP/1.1 503 Service Unavailable  
Retry-After: 120
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 503 for client-specific throttling when 429 is more precise. Retry storms caused by ignoring backoff and Retry-After.

Compare with: 429 Too Many Requests | 502 Bad Gateway | 504 Gateway Timeout

504 Gateway Timeout

504 Gateway Timeout

A gateway or proxy did not receive a timely response from the upstream server needed to complete the request.

Profile	Very common in distributed systems	Retry lens	Retry with care/backoff
Response body	Minimal/problem details	Typical methods	Any through intermediary
Reference	RFC 9110, Section 15.6.5		

When it appears

- API gateway waits too long for an upstream service.
- The request reached an intermediary; the upstream response did not arrive within its timeout budget.

Wire example

```
GET /recommendations HTTP/1.1
```

```
HTTP/1.1 504 Gateway Timeout
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Confusing 504 with 408. 408 concerns receiving the client request; 504 concerns a gateway waiting for an upstream response.

Compare with: 408 Request Timeout | 502 Bad Gateway | 503 Service Unavailable

505 HTTP Version Not Supported

505 HTTP Version Not Supported

The server does not support, or refuses to support, the major HTTP version used by the request.

Profile	Rare	Retry lens	Use a supported HTTP version
Response body	Problem details optional	Typical methods	Any
Reference	RFC 9110, Section 15.6.6		

When it appears

- Protocol version is unsupported at the server boundary.

Wire example

```
GET / HTTP/9.9
```

```
HTTP/1.1 505 HTTP Version Not Supported
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 505 for unsupported API versions such as /v9; that is an application routing/versioning concern, not necessarily HTTP-version support.

Compare with: 426 Upgrade Required | 501 Not Implemented

506 Variant Also Negotiates

506 Variant Also Negotiates

A configuration error in transparent content negotiation caused the selected variant itself to participate in negotiation.

Profile	Very rare	Retry lens	Server configuration must be fixed
Response body	Problem details optional	Typical methods	GET/HEAD
Reference	RFC 2295		

When it appears

- Negotiated variant configuration recursively points back into negotiation.

Wire example

```
GET /resource HTTP/1.1
Accept: text/html
```

```
HTTP/1.1 506 Variant Also Negotiates
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 506 as a generic “variant not found” response.

Compare with: 500 Internal Server Error

507 Insufficient Storage

507 Insufficient Storage

The server cannot store the representation needed to complete the request.

Profile	Rare / WebDAV	Retry lens	Retry after storage/capacity issue changes
Response body	Problem details useful	Typical methods	PUT, POST, WebDAV methods
Reference	RFC 4918		

When it appears

- WebDAV operation cannot persist required state because storage is insufficient.

Wire example

```
PUT /dav/archive.zip HTTP/1.1
Content-Length: 1000000000
```

```
HTTP/1.1 507 Insufficient Storage
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 507 as a universal disk-full status for every web application without documenting semantics.

Compare with: 413 Content Too Large | 503 Service Unavailable

508 Loop Detected

508 Loop Detected

The server detected an infinite loop while processing a WebDAV request.

Profile	Rare / WebDAV	Retry lens	Fix binding/loop state
Response body	Problem details / DAV response	Typical methods	WebDAV methods
Reference	RFC 5842		

When it appears

- Binding traversal causes a loop while processing a request such as depth traversal.

Wire example

```
PROPFIND /dav/root HTTP/1.1
Depth: infinity
```

```
HTTP/1.1 508 Loop Detected
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Confusing it with generic redirect loops in browsers.

Compare with: 208 Already Reported

510 Not Extended

510 Not Extended

An obsolete experimental status from an older HTTP extension framework.

Profile	Obsoleted	Retry lens	Do not use in new designs
Response body	Not applicable	Typical methods	Historical
Reference	RFC 2774 (obsoleted)		

When it appears

- Retained in the registry with an obsoleted marker for historical reference.
- New APIs should not assign fresh semantics to 510.

Wire example

```
[historical extension request]
```

```
HTTP/1.1 510 Not Extended
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 510 in new application protocols.

Compare with: 501 Not Implemented

511 Network Authentication Required

511 Network Authentication Required

The client needs to authenticate to gain network access, typically through a captive portal.

Profile	Specialized	Retry lens	Authenticate to network, then retry
Response body	Portal metadata may be supplied	Typical methods	Any
Reference	RFC 6585		

When it appears

- Wi-Fi hotspot or access network intercepts traffic before normal origin access.
- Intended for network access authentication, not origin-server application login.

Wire example

```
GET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 511 Network Authentication Required
Content-Type: text/html

[network login portal information]
```

QA lens

- Identify which component generated the status: origin, reverse proxy, API gateway, load balancer, or captive network.
- Verify retry/backoff behavior for operations where retry is safe.
- Confirm observability includes correlation/trace identifiers without exposing internal details to clients.

Common misuse. Using 511 instead of 401 for application authentication.

Compare with: 401 Unauthorized | 407 Proxy Authentication Required

Unused, Obsolete, Temporary, and Unassigned Numbers

Why these labels matter

A three-digit number is not automatically a standardized HTTP status code. The registry distinguishes assigned status codes from unassigned ranges, and it can mark particular values as unused, temporary, or obsolete. Applications should not invent semantics inside an unassigned range and then assume interoperability.

Label	Meaning in this reference
Temporary	Registered for a limited period while a specification evolves. Re-check the registry before implementation/publication.
Unused	A particular number is intentionally shown without active HTTP semantics.
Obsolete	Historical semantics exist, but new designs should not depend on them.
Unassigned	No standardized status is assigned to that number/range in the registry.

Current unassigned ranges

Range	Registry status
105-199	Unassigned
209-225	Unassigned
227-299	Unassigned
309-399	Unassigned
419-420	Unassigned
427	Unassigned
430	Unassigned
432-450	Unassigned
452-499	Unassigned
509	Unassigned
512-599	Unassigned

Non-standard ecosystem codes. Servers, CDNs, reverse proxies, frameworks, and vendors sometimes expose additional three-digit codes. Those can be operationally useful, but they should be documented as vendor-specific rather than presented as IANA-standard HTTP status codes.

PART 9

The Confusing Codes - Decision Guide

This part is intentionally comparative. Definitions are useful, but production mistakes usually happen because two nearby codes both seem plausible. Each page below asks for the distinguishing decision.

200 vs 201 vs 202 vs 204

Decision question. Choose by what happened, not by “success” alone.

Choice	Use when
200 OK	Work completed and a representation/result is being returned.
201 Created	Work completed and created one or more resources.
202 Accepted	Work was accepted but completion is still pending.
204 No Content	Work completed and there is no response content to send.

Fast rule. Ask: Did work finish? Was a resource created? Is there content to return?

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

201 vs 202

Decision question. Creation completed now vs work accepted for later completion.

Choice	Use when
201 Created	The resource already exists as a result of this request.
202 Accepted	The request is queued/accepted; final success is not yet known.

Fast rule. If the client must poll a job or wait for later completion, 202 is usually the clearer signal.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

204 vs 205

Decision question. Both avoid a response representation, but only one instructs the user agent to reset its view.

Choice	Use when
204 No Content	Success, no content.
205 Reset Content	Success, no content, and reset the document view that initiated the request.

Fast rule. Use 205 only when the reset instruction is part of the intended semantics.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

301 vs 308

Decision question. Both are permanent redirects; method preservation is the differentiator.

Choice	Use when
301 Moved Permanently	Permanent, but historical user-agent behavior can rewrite non-GET methods.
308 Permanent Redirect	Permanent and explicitly preserves method/body on redirection.

Fast rule. For permanent POST/PUT/PATCH endpoint migration where method preservation matters, 308 is clearer.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

302 vs 303 vs 307

Decision question. Three temporary redirect behaviors that are often mixed up.

Choice	Use when
302 Found	Temporary location; method handling follows longstanding user-agent behavior.
303 See Other	Retrieve another URI using GET/HEAD; ideal for Post/Redirect/Get.
307 Temporary Redirect	Temporary location while preserving the original method and request content.

Fast rule. Ask whether the next request should be GET or should repeat the original method.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

304 vs 204

Decision question. Both commonly have no response content, but they mean completely different things.

Choice	Use when
304 Not Modified	Conditional GET/HEAD may reuse cached representation.
204 No Content	The operation succeeded and no response content is being sent.

Fast rule. 304 is cache validation, not a generic empty success.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

400 vs 422

Decision question. Parsing/structural request problem vs understood representation with unprocessable instructions.

Choice	Use when
400 Bad Request	The request cannot be processed as a valid request, such as malformed JSON or invalid general syntax.
422 Unprocessable Content	Content type and syntax are understood, but semantic/domain validation fails.

Fast rule. Example: broken JSON -> 400; valid JSON with invalid email/domain value -> often 422.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

401 vs 403

Decision question. Authentication vs authorization.

Choice	Use when
401 Unauthorized	Valid authentication is missing or not accepted.
403 Forbidden	The server understands the request but refuses it, commonly because the authenticated principal lacks permission.

Fast rule. Remember: 401 asks “who are you / can credentials be accepted?”; 403 asks “may this requester do this?”

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

403 vs 404

Decision question. Access denied vs resource not found/undisclosed.

Choice	Use when
403 Forbidden	The server refuses access and may acknowledge the target.
404 Not Found	The server cannot find the target or intentionally chooses not to disclose its existence.

Fast rule. Security policy may intentionally use 404 to avoid resource enumeration.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

404 vs 410

Decision question. Unknown/not currently found vs known permanent removal.

Choice	Use when
404 Not Found	No current resource is available; permanence may be unknown.
410 Gone	The resource is gone and the condition is expected to be permanent.

Fast rule. Use 410 when permanence is known and useful to clients/crawlers.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

405 vs 501

Decision question. Unsupported on this resource vs unsupported by the server implementation.

Choice	Use when
405 Method Not Allowed	Method is recognized, but not allowed for this target; include Allow.
501 Not Implemented	The server does not implement the functionality needed to fulfill the request, often an unsupported method.

Fast rule. If GET works but DELETE is prohibited on that route, think 405. If the server does not implement an unfamiliar method at all, think 501.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

408 vs 504

Decision question. Timeout at different sides of an intermediary boundary.

Choice	Use when
408 Request Timeout	The server timed out receiving the client request.
504 Gateway Timeout	A gateway/proxy timed out waiting for an upstream response.

Fast rule. Ask which hop was waiting for which other hop.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

409 vs 412

Decision question. General resource-state conflict vs explicit HTTP precondition failure.

Choice	Use when
409 Conflict	Current resource state conflicts with the requested operation.
412 Precondition Failed	A supplied condition such as If-Match evaluated false.

Fast rule. When an HTTP conditional header specifically failed, 412 is the precise signal.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

409 vs 422

Decision question. Resource state conflict vs semantic problem inside otherwise understandable content.

Choice	Use when
409 Conflict	The request clashes with current server/resource state.
422 Unprocessable Content	The representation is understood but its instructions/values cannot be processed.

Fast rule. Validation of a field value often points to 422; an edit/version or state-transition collision often points to 409.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

413 vs 414 vs 431

Decision question. Three different things are “too large.”

Choice	Use when
413 Content Too Large	Request body/content is too large.
414 URI Too Long	Request target/URI is too long.
431 Request Header Fields Too Large	Header fields are too large.

Fast rule. Identify the oversized part of the HTTP message.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

406 vs 415

Decision question. Response negotiation vs request representation.

Choice	Use when
406 Not Acceptable	Server cannot produce what the client says it will accept.
415 Unsupported Media Type	Server cannot process the media type/encoding the client sent.

Fast rule. Accept -> 406 side; Content-Type -> 415 side.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

428 vs 412

Decision question. Missing required condition vs supplied condition failed.

Choice	Use when
428 Precondition Required	Server requires a conditional request but the client did not supply the required precondition.
412 Precondition Failed	Client supplied a precondition and it evaluated false.

Fast rule. 428 tells the client to add the guard; 412 tells the client the guard did not match.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

429 vs 503

Decision question. Client-specific rate policy vs service-wide temporary unavailability.

Choice	Use when
429 Too Many Requests	This client/key/IP/resource exceeded an applicable request rate.
503 Service Unavailable	The service is temporarily unable to handle requests, often maintenance or overload.

Fast rule. Both can use Retry-After, but the scope of the problem is different.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

500 vs 502 vs 503 vs 504

Decision question. The four server-side codes most often blurred together.

Choice	Use when
500 Internal Server Error	This server encountered an unexpected failure.
502 Bad Gateway	Gateway/proxy received an invalid upstream response.
503 Service Unavailable	Service is temporarily unable to handle the request.
504 Gateway Timeout	Gateway/proxy waited too long for upstream.

Fast rule. Draw the request path. The component that fails and the nature of the failure usually identifies the right code.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

501 vs 505

Decision question. Unsupported functionality vs unsupported HTTP protocol version.

Choice	Use when
501 Not Implemented	Required method/functionality is not implemented.
505 HTTP Version Not Supported	The HTTP major version is unsupported/refused.

Fast rule. API versioning such as /v2 vs /v3 is not the same thing as HTTP protocol versioning.

QA exercise

Build one positive test for each choice and one “near miss” where the neighboring code would be tempting but wrong. Validate not only the number, but also headers, body rules, retry behavior, and the component that generated the response.

The Confusing Methods - Design Guide

POST vs PUT

	POST	PUT
Target URI	Usually names the processor/collection; server often chooses a created resource URI.	Client targets the resource URI whose state is being created/replaced.
Idempotent	No (by registered semantics).	Yes.
Typical result	200, 201, 202, 204.	200, 201, 204.
Design clue	“Process this representation according to the target’s semantics.”	“Make this target resource have this representation/state.”

Do not reduce the distinction to “POST = create, PUT = update.” Both can create; the resource targeting and semantics are the deeper distinction.

PUT vs PATCH

PUT commonly represents replacement of target resource state; PATCH carries instructions describing changes. PUT is registered idempotent. PATCH is not inherently idempotent, although a specific patch format/document can be designed so repeating it is safe.

```
PUT /profile/42
{"name":"Mira","city":"Bengaluru","timezone":"Asia/Kolkata"}
```

```
PATCH /profile/42
{"city":"Mysuru"}
```

GET vs HEAD

HEAD asks for the metadata that GET would produce without transferring the response content. Use it for metadata checks, validators, and lightweight existence/capability checks where supported. A server should not invent unrelated HEAD semantics.

OPTIONS vs HEAD

HEAD asks “what would GET metadata look like without the content?” OPTIONS asks “what communication options/capabilities apply here?” CORS preflight is an OPTIONS use case, not a HEAD use case.

Safe vs idempotent vs retry-safe

Safe and idempotent are protocol properties. Retry-safe is a system decision that also depends on timeouts, duplicate detection, side effects, idempotency keys, payment/workflow semantics, and whether the client knows if the first attempt reached the server.

Example: DELETE is idempotent by method semantics, but a retry can still encounter a different response such as 404 after an earlier successful deletion. “Same intended effect” does not mean “same response.”

HTTP for API Testing and Debugging

A status-code assertion is not enough

For each API operation, verify the complete contract:

- method and request target;
- authentication and authorization behavior;
- request Content-Type and response content negotiation;
- status code and whether that code accurately describes the outcome;
- required/meaningful headers such as Location, Allow, WWW-Authenticate, Retry-After, ETag, Content-Range;
- response-body presence/absence and media type;
- problem details, field-level validation, and correlation identifiers;
- caching, conditional requests, concurrency, and retry behavior;
- behavior through gateways/proxies as well as direct origin behavior.

Negative test families

Test family	Examples
Malformed request	Invalid JSON, broken framing, illegal request target -> often 400.
Authentication	Missing, expired, malformed credential -> 401; proxy challenge -> 407.
Authorization	Valid identity without permission -> 403; possibly 404 when policy hides existence.
Representation	Unsupported Content-Type -> 415; unacceptable response type -> 406.
Validation	Well-formed content with invalid domain values -> often 422.
State/conflict	Stale state, duplicate transition, incompatible resource state -> often 409/412.
Limits	Body -> 413; URI -> 414; headers -> 431; rate -> 429.
Infrastructure	Unexpected origin fault -> 500; invalid upstream -> 502; unavailable -> 503; upstream timeout -> 504.

Debug 5xx by drawing the hop chain

When a client sees a 5xx code, identify which component actually generated it. A browser may call a CDN, which calls a WAF, which calls an API gateway, which calls a service mesh proxy, which calls an application, which calls another service. 500/502/503/504 become far easier to distinguish once the responding hop and upstream relationship are visible.

Retry decision checklist

Before retrying, ask:

1. Is the method safe or idempotent by semantics?
2. If not, does the API support an idempotency key or duplicate-suppression mechanism?
3. Did the server provide Retry-After?
4. Could the first attempt have succeeded even though the client did not receive the response?
5. Will retries amplify overload (retry storm)?
6. Is exponential backoff with jitter appropriate?

Contract-review smell: “always return 200”

An API that encodes every outcome inside a 200 response body gives up valuable protocol semantics. Gateways, clients, observability tools, caches, retries, and generic HTTP tooling cannot reason correctly from the status line. A domain error can still have rich body detail, but the HTTP status should communicate the transport/application boundary outcome accurately.

Which Status Code Fits This Situation?

Situation	Start with	Check also
Resource returned successfully	200	206 / 304 for range/cache cases
Resource created now	201	Location header
Work accepted but still processing	202	Job/status resource
Successful operation, nothing to return	204	205 only if reset semantics matter
Malformed request	400	415 / 422 when more precise
Authentication missing/invalid	401	WWW-Authenticate
Authenticated but not permitted	403	404 when disclosure policy hides existence
Resource not found	404	410 if permanently gone
Known method not allowed here	405	Allow header; compare 501
Unacceptable response representation	406	Accept headers
Client request receive timeout	408	Compare 504
Current resource state conflicts	409	412 / 422 depending on cause
HTTP precondition supplied and failed	412	428 if condition was required but missing
Request body too large	413	414 / 431
URI too long	414	Query-string design
Unsupported request media type	415	Compare 406 / 422
Range cannot be satisfied	416	Content-Range
Semantically invalid but understandable content	422	Compare 400 / 409
Required conditional request missing	428	Add If-Match or required guard
Rate limit exceeded	429	Retry-After; compare 503
Unexpected server failure	500	Trace/correlation ID
Server does not implement required functionality	501	Compare 405
Gateway received invalid upstream response	502	Identify upstream
Temporary service overload/maintenance	503	Retry-After
Gateway timed out waiting upstream	504	Compare 408

Master Status Code Table

Code	Name	Profile	Reference
100	Continue	Specialized	RFC 9110, Section 15.2.1
101	Switching Protocols	Specialized	RFC 9110, Section 15.2.2
102	Processing	Rare / WebDAV	RFC 2518
103	Early Hints	Specialized	RFC 8297
104	Upload Resumption Supported	Temporary / experimental	draft-ietf-httpbis-resumable-upload-05 (temporary IANA registration)
200	OK	Very common	RFC 9110, Section 15.3.1
201	Created	Very common	RFC 9110, Section 15.3.2
202	Accepted	Common	RFC 9110, Section 15.3.3
203	Non-Authoritative Information	Rare	RFC 9110, Section 15.3.4
204	No Content	Very common	RFC 9110, Section 15.3.5
205	Reset Content	Rare	RFC 9110, Section 15.3.6
206	Partial Content	Common in media/-downloads	RFC 9110, Section 15.3.7
207	Multi-Status	Rare / WebDAV	RFC 4918
208	Already Reported	Rare / WebDAV	RFC 5842
226	IM Used	Very rare	RFC 3229
300	Multiple Choices	Uncommon	RFC 9110, Section 15.4.1
301	Moved Permanently	Very common	RFC 9110, Section 15.4.2
302	Found	Very common	RFC 9110, Section 15.4.3
303	See Other	Common	RFC 9110, Section 15.4.4
304	Not Modified	Very common in caching	RFC 9110, Section 15.4.5
305	Use Proxy	Deprecated / avoid	RFC 9110, Section 15.4.6
306	Unused	Unused	RFC 9110, Section 15.4.7
307	Temporary Redirect	Common	RFC 9110, Section 15.4.8
308	Permanent Redirect	Common	RFC 9110, Section 15.4.9
400	Bad Request	Very common	RFC 9110, Section 15.5.1
401	Unauthorized	Very common	RFC 9110, Section 15.5.2
402	Payment Required	Reserved / application-specific	RFC 9110, Section 15.5.3
403	Forbidden	Very common	RFC 9110, Section 15.5.4
404	Not Found	Very common	RFC 9110, Section 15.5.5
405	Method Not Allowed	Common	RFC 9110, Section 15.5.6
406	Not Acceptable	Uncommon	RFC 9110, Section 15.5.7
407	Proxy Authentication Required	Specialized	RFC 9110, Section 15.5.8
408	Request Timeout	Common at infrastructure edge	RFC 9110, Section 15.5.9
409	Conflict	Very common	RFC 9110, Section 15.5.10
410	Gone	Uncommon but useful	RFC 9110, Section 15.5.11
411	Length Required	Rare	RFC 9110, Section 15.5.12
412	Precondition Failed	Common in concurrency/caching	RFC 9110, Section 15.5.13
413	Content Too Large	Common	RFC 9110, Section 15.5.14
414	URI Too Long	Uncommon	RFC 9110, Section 15.5.15
415	Unsupported Media Type	Very common	RFC 9110, Section 15.5.16
416	Range Not Satisfiable	Common in range-capable resources	RFC 9110, Section 15.5.17
417	Expectation Failed	Rare	RFC 9110, Section 15.5.18
418	Unused	Unused	RFC 9110, Section 15.5.19
421	Misdirected Request	Specialized	RFC 9110, Section 15.5.20
422	Unprocessable Content	Very common in modern APIs	RFC 9110, Section 15.5.21
423	Locked	Rare / WebDAV	RFC 4918
424	Failed Dependency	Rare / WebDAV	RFC 4918
425	Too Early	Specialized	RFC 8470
426	Upgrade Required	Specialized	RFC 9110, Section 15.5.22

Code	Name	Profile	Reference
428	Precondition Required	Common in concurrency-aware APIs	RFC 6585
429	Too Many Requests	Very common	RFC 6585
431	Request Header Fields Too Large	Uncommon	RFC 6585
451	Unavailable For Legal Reasons	Specialized	RFC 7725
500	Internal Server Error	Very common	RFC 9110, Section 15.6.1
501	Not Implemented	Uncommon	RFC 9110, Section 15.6.2
502	Bad Gateway	Very common in distributed systems	RFC 9110, Section 15.6.3
503	Service Unavailable	Very common	RFC 9110, Section 15.6.4
504	Gateway Timeout	Very common in distributed systems	RFC 9110, Section 15.6.5
505	HTTP Version Not Supported	Rare	RFC 9110, Section 15.6.6
506	Variant Also Negotiates	Very rare	RFC 2295
507	Insufficient Storage	Rare / WebDAV	RFC 4918
508	Loop Detected	Rare / WebDAV	RFC 5842
510	Not Extended	Obsoleted	RFC 2774 (obsoleted)
511	Network Authentication Required	Specialized	RFC 6585

Glossary

Cache validator

Metadata such as an ETag or modification date used to decide whether a cached representation is still valid.

Conditional request

A request whose effect depends on one or more preconditions such as If-Match or If-None-Match.

Gateway

An intermediary that receives an HTTP request and acts on it by communicating with another server to produce a response.

Idempotent

Multiple identical requests have the same intended effect as one request.

Intermediary

A component between client and origin, such as a proxy, gateway, cache, or tunnel endpoint.

Media type

A label such as application/json that describes the format of representation data.

Origin server

The server that is authoritative for the target resource according to the request routing context.

Representation

Information intended to reflect a past, current, or desired state of a resource, carried as representation metadata plus data.

Safe method

A method whose defined semantics are essentially read-only from the client's requested intent.

Status class

The first digit grouping: 1xx informational, 2xx success, 3xx redirection, 4xx client error, 5xx server error.

Target resource

The resource identified by the request target after URI interpretation/routing.

Official References

- IANA HTTP Status Code Registry: <https://www.iana.org/assignments/http-status-codes>
- IANA HTTP Method Registry: <https://www.iana.org/assignments/http-methods>
- RFC 9110, HTTP Semantics: <https://www.rfc-editor.org/rfc/rfc9110>
- RFC 9111, HTTP Caching: <https://www.rfc-editor.org/rfc/rfc9111>
- RFC 9112, HTTP/1.1: <https://www.rfc-editor.org/rfc/rfc9112>
- RFC 9113, HTTP/2: <https://www.rfc-editor.org/rfc/rfc9113>
- RFC 9114, HTTP/3: <https://www.rfc-editor.org/rfc/rfc9114>
- RFC 9457, Problem Details for HTTP APIs: <https://www.rfc-editor.org/rfc/rfc9457>
- Extension RFCs are cited on the relevant method/status pages and are discoverable through the IANA registries above.

Index

Accepted (202), 54
Already Reported (208), 60

Bad Gateway (502), 105
Bad Request (400), 73

Conflict (409), 82
Content Too Large (413), 86
Continue (100), 46
Created (201), 53

Early Hints (103), 49
Expectation Failed (417), 90

Failed Dependency (424), 95
Forbidden (403), 76
Found (302), 65

Gateway Timeout (504), 107
Gone (410), 83

HTTP method
 ACL, 4
 ASTERISK (*), 44
 BASELINE-CONTROL, 5
 BIND, 6
 CHECKIN, 7
 CHECKOUT, 8
 CONNECT, 9
 COPY, 10
 DELETE, 11
 GET, 12
 HEAD, 13
 LABEL, 14
 LINK, 15
 LOCK, 16
 MERGE, 17
 MKACTIVITY, 18
 MKCALENDAR, 19
 MKCOL, 20
 MKREDIRECTREF, 21
 MKWORKSPACE, 22
 MOVE, 23
 OPTIONS, 24
 ORDERPATCH, 25
 PATCH, 26
 POST, 27
 PRI, 28
 PROPFIND, 29
 PROPPATCH, 30
 PUT, 31
 QUERY, 32
 REBIND, 33
 REPORT, 34
 SEARCH, 35
 TRACE, 36
 UNBIND, 37
 UNCHECKOUT, 38
 UNLINK, 39
 UNLOCK, 40

 UPDATE, 41
 UPDATEREDIRECTREF, 42
 VERSION-CONTROL, 43
HTTP status code
 100 Continue, 46
 101 Switching Protocols, 47
 102 Processing, 48
 103 Early Hints, 49
 104 Upload Resumption Supported, 50
 200 OK, 52
 201 Created, 53
 202 Accepted, 54
 203 Non-Authoritative Information, 55
 204 No Content, 56
 205 Reset Content, 57
 206 Partial Content, 58
 207 Multi-Status, 59
 208 Already Reported, 60
 226 IM Used, 61
 300 Multiple Choices, 63
 301 Moved Permanently, 64
 302 Found, 65
 303 See Other, 66
 304 Not Modified, 67
 305 Use Proxy, 68
 306 Unused, 69
 307 Temporary Redirect, 70
 308 Permanent Redirect, 71
 400 Bad Request, 73
 401 Unauthorized, 74
 402 Payment Required, 75
 403 Forbidden, 76
 404 Not Found, 77
 405 Method Not Allowed, 78
 406 Not Acceptable, 79
 407 Proxy Authentication Required, 80
 408 Request Timeout, 81
 409 Conflict, 82
 410 Gone, 83
 411 Length Required, 84
 412 Precondition Failed, 85
 413 Content Too Large, 86
 414 URI Too Long, 87
 415 Unsupported Media Type, 88
 416 Range Not Satisfiable, 89
 417 Expectation Failed, 90
 418 Unused, 91
 421 Misdirected Request, 92
 422 Unprocessable Content, 93
 423 Locked, 94
 424 Failed Dependency, 95
 425 Too Early, 96
 426 Upgrade Required, 97
 428 Precondition Required, 98
 429 Too Many Requests, 99
 431 Request Header Fields Too Large, 100
 451 Unavailable For Legal Reasons, 101
 500 Internal Server Error, 103

- 501 Not Implemented, 104
- 502 Bad Gateway, 105
- 503 Service Unavailable, 106
- 504 Gateway Timeout, 107
- 505 HTTP Version Not Supported, 108
- 506 Variant Also Negotiates, 109
- 507 Insufficient Storage, 110
- 508 Loop Detected, 111
- 510 Not Extended, 112
- 511 Network Authentication Required, 113
- HTTP Version Not Supported (505), 108
- IM Used (226), 61
- Insufficient Storage (507), 110
- Internal Server Error (500), 103
- Length Required (411), 84
- Locked (423), 94
- Loop Detected (508), 111
- Method Not Allowed (405), 78
- Misdirected Request (421), 92
- Moved Permanently (301), 64
- Multi-Status (207), 59
- Multiple Choices (300), 63
- Network Authentication Required (511), 113
- No Content (204), 56
- Non-Authoritative Information (203), 55
- Not Acceptable (406), 79
- Not Extended (510), 112
- Not Found (404), 77
- Not Implemented (501), 104
- Not Modified (304), 67
- OK (200), 52
- Partial Content (206), 58
- Payment Required (402), 75
- Permanent Redirect (308), 71
- Precondition Failed (412), 85
- Precondition Required (428), 98
- Processing (102), 48
- Proxy Authentication Required (407), 80
- Range Not Satisfiable (416), 89
- Request Header Fields Too Large (431), 100
- Request Timeout (408), 81
- Reset Content (205), 57
- See Other (303), 66
- Service Unavailable (503), 106
- Status code comparison
 - 200 vs 201 vs 202 vs 204, 116
 - 201 vs 202, 117
 - 204 vs 205, 118
 - 301 vs 308, 119
 - 302 vs 303 vs 307, 120
 - 304 vs 204, 121
 - 400 vs 422, 122
 - 401 vs 403, 123
 - 403 vs 404, 124
 - 404 vs 410, 125
 - 405 vs 501, 126
 - 406 vs 415, 131
 - 408 vs 504, 127
 - 409 vs 412, 128
 - 409 vs 422, 129
 - 413 vs 414 vs 431, 130
 - 428 vs 412, 132
 - 429 vs 503, 133
 - 500 vs 502 vs 503 vs 504, 134
 - 501 vs 505, 135
- Switching Protocols (101), 47
- Temporary Redirect (307), 70
- Too Early (425), 96
- Too Many Requests (429), 99
- Unauthorized (401), 74
- Unavailable For Legal Reasons (451), 101
- Unprocessable Content (422), 93
- Unsupported Media Type (415), 88
- Unused (306), 69
- Unused (418), 91
- Upgrade Required (426), 97
- Upload Resumption Supported (104), 50
- URI Too Long (414), 87
- Use Proxy (305), 68
- Variant Also Negotiates (506), 109