API Guideline Wasserstoffmarkt: Teil 1 – Regelungen zur Nutzung und Erstellung von API-Webdiensten

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface / Anwendungsschnittstelle
BNetzA	Bundesnetzagentur
ECMA	European Computer Manufacturers Association / Organisation für die Standardisierung von JavaScript
etc	Et cetera / und so weiter
ggf	gegebenenfalls
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronic Engineers
IP	Internet Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
RFC	Request for Comments / technische Dokumente und Standards der Internet Engineering Taskforce
SM-PKI	Smart Meter Public Key Infrastructure
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
WaKandA	Wasserstoff Kapazitäten Grundmodell und Abwicklung des Netzzugangs
WasABi	Wasserstoff Ausgleichs- und Bilanzierungsmodell
z.B.	zum Beispiel

Teil 1
–
Regelungen zur Nutzung und Erstellung von API-Webdiensten

1 Einleitung

1.1 Ziel des Dokuments

Dieses Dokument definiert verbindliche fachliche und technische Vorgaben für die Erstellung und Nutzung von API-Webdiensten im regulierten Wasserstoffmarkt.

Es beschreibt die Prinzipien, nach denen API-basierte Kommunikationsprozesse gemäß der Festlegungen WasABi [BNetzA BK7-24-01-014] und WaKandA [BNetzA BK7-24-01-015] umzusetzen sind.

Die in diesem Dokument beschriebenen Anforderungen MÜSSEN von allen Marktrollen eingehalten werden, die API-Webdienste im Rahmen der regulierten Marktprozesse bereitstellen oder nutzen.

Hinweis:

Diese Guideline orientiert sich in einzelnen strukturellen und sprachlichen Bestandteilen an der BDEW API-Guideline Version 1.0a [BDEW API 1.0a].

1.2 Motivation und Hintergrund

Mit den BNetzA-Festlegungen WasABi und WaKandA wurden verbindliche Prozessvorgaben für die Austauschbilanzierung im Wasserstoffmarkt geschaffen. Diese Vorgaben verlangen für bestimmte Prozessschritte die Nutzung standardisierter API-Webdienste, um eine konsistente, interoperable und automatisierte Kommunikation zwischen den Marktrollen sicherzustellen.

Eine gemeinsame Guideline ist erforderlich, um:

- einheitliche URL-Strukturen, Methoden und Datenobjekte
- konsistente Datentypen und JSON-Standards
- abgestimmte Regeln für Versionierung und Änderungsmanagement
- klare Vorgaben für Statuscodes, Fehlerverhalten und Resilienz

zu gewährleisten.

Die vorliegende Guideline beschreibt daher die grundlegenden Design- und Implementierungsprinzipien, die den Rahmen für alle API-Webdienste im regulierten Wasserstoffmarkt bilden.

1.3 Ziel der Guideline und Geltungsbereich

Ziel dieser Guideline ist es, konsistente, eindeutige und interoperable Web-API-Schnittstellen zu definieren, die für alle WasABi und WaKandA relevanten Marktprozesse eingesetzt werden. Die Guideline beschreibt Mindestanforderungen, die jede API erfüllen MUSS, um einen sicheren und zuverlässigen elektronischen Datenaustausch zu gewährleisten.

Der Geltungsbereich umfasst insbesondere:

- alle API-Webdienste, die im Rahmen der Festlegung WasABi und WaKandA bereitgestellt, genutzt oder weiterentwickelt werden,
- alle relevanten fachlichen Objekte, Transportstrukturen und JSON-Spezifikationen,
- alle Marktrollen, die an den betroffenen Prozessen teilnehmen.

Dieses Dokument benennt nicht die ggf. bestehenden rechtlichen Folgen, die entstehen können, wenn aufgrund eines abweichenden Vorgehens kein gesicherter elektronischer Datenaustausch gewährleistet ist.

2 Grundlagen und Terminologie

2.1 Schlüsselwörter in der API-Guideline

Die Schlüsselwörter „MUSS“/„MÜSSEN“ (MUST), „DARF NICHT“ (MUST NOT), „ERFORDERLICH“ (REQUIRED), „SOLL“ (SHALL), „SOLL NICHT“ (SHALL NOT), „SOLLTE“ (SHOULD), „SOLLTE NICHT“ (SHOULD NOT), „EMPFOHLEN“ (RECOMMENDED), „NICHT EMPFOHLEN“ (NOT RECOMMENDED), „KANN/DARF“ (MAY) und „OPTIONAL“ (OPTIONAL) sind gemäß [RFC2119] und [RFC8174] zu interpretieren, wenn sie in Großbuchstaben verwendet werden.

2.2 Glossar

Begriff	Beschreibung
API-Anbieter	Bietet einen Webservice an, über den die beschriebene API genutzt werden kann.
API-Nutzer	Nutzt als Client mittels eines angebotenen Webservice die beschriebene API.
Kommunikationsendpunkt	URL und Port (URI) des API-Webservice
Data Hub	Zentrale technische Kommunikations- und Datenaustauschplattform

2.3 Marktrollen und Kommunikationsbeziehungen

2.3.1 Grundsatz

Für die in WasABi und WaKandA definierten, regulierten Prozesse erfolgt die technische Marktkommunikation ausschließlich über den Data Hub; eine unmittelbare technische Kommunikation zwischen den Marktrollen ist nicht vorgesehen. Die fachliche und rechtliche Verantwortung verbleibt ungeachtet des zentralisierten Kommunikationsweges bei der jeweils zuständigen Marktrolle.

2.3.2 Marktrollen

Die nachfolgenden Marktrollen bilden die Grundlage für die Zuordnung von Aufgaben, Verantwortlichkeiten und Kommunikationsbeziehungen im Rahmen dieser Guideline.

- Data Hub
  Der Data Hub ist die zentrale technische Kommunikations- und Datenaustauschplattform des Wasserstoffmarktes.
- Marktgebietsverantwortlicher
  Der Marktgebietsverantwortliche ist die von allen Wasserstoff-Transportnetzbetreibern

gemeinsam benannte juristische Person, die für den betrieblichen, bilanziellen und organisatorischen Ausgleich des gesamten nationalen Wasserstoff-Marktgebiets verantwortlich ist. Er fungiert damit als zentrale Koordinations- und Steuerungsstelle für das deutsche Wasserstoffnetz.

- Netzbetreiber
  Ein Netzbetreiber im Wasserstoffmarkt ist eine natürliche oder juristische Person, die ein Wasserstoffnetz betreibt und für dessen sicheren, diskriminierungsfreien Betrieb, Wartung sowie gegebenenfalls den Ausbau verantwortlich ist und den regulierten Netzzugang nach den Vorgaben des Energiewirtschaftsgesetzes gewährleistet.
- Bilanzkreisverantwortlicher
  Bilanzkreisverantwortlicher im Wasserstoffmarkt ist eine natürliche oder juristische Person, die gegenüber dem Marktgebietsverantwortlichen für die ausgeglichene Bilanz eines Wasserstoff-Bilanzkreises verantwortlich ist und die Einspeise- und Ausspeisemengen bilanziell koordiniert.
- Transportkunde
  Ein Transportkunde ist jede natürliche oder juristische Person, die auf Grundlage eines Netzzugangs- bzw. Transportvertrags Kapazität in einem Wasserstoffnetz gebucht hat und diese zur Ein- oder Ausspeisung von Wasserstoff nutzt. Der Transportkunde ist damit Nutzer der Transportleistung des Wasserstoffnetzes.
- Speicherbetreiber
  Ein Speicherbetreiber im Wasserstoffmarkt ist eine natürliche oder juristische Person, die eine Wasserstoffspeicheranlage betreibt und für deren sicheren, diskriminierungsfreien Betrieb sowie die Bereitstellung von Speicherkapazitäten verantwortlich ist. Er ermöglicht im Rahmen der geltenden regulatorischen Vorgaben die Einspeicherung, Speicherung und Ausspeicherung von Wasserstoff.

2.3.3 Kommunikationsbeziehungen

(1) Die technische Kommunikation zwischen den Marktrollen erfolgt ausschließlich über den Data Hub.

(2) Eine direkte technische Marktkommunikation zwischen Marktgebietsverantwortlichem, Bilanzkreisverantwortlichem, Transportkunde, Netzbetreiber, Speicherbetreibern und weiteren Marktrollen ist für die in dieser Guideline geregelten Prozesse nicht vorgesehen.

(3) Der Data Hub stellt zentral eine durchgängige Berechtigungssicherung sicher und ermöglicht eine feingranulare Steuerung der Zugriffe auf Daten und Funktionen. Dies setzt ein konsistentes Rollen- und Rechtemodell voraus. Die Authentifizierung der Kommunikationspartner und die technische Grundlage der Berechtigungssicherung erfolgen zertifikatsbasiert.

Die Zentralisierung des Kommunikationsweges über den Data Hub berührt nicht die fachliche, operative oder vertragliche Verantwortung der jeweils zuständigen Marktrolle. Insbesondere verbleibt die inhaltliche Verantwortung für Prozesse, die über den Data Hub abgewickelt werden, bei der jeweils zuständigen Marktrolle.

3 Anforderungen an den Betrieb

Die nachfolgenden Anforderungen regeln den sicheren, stabilen, nachvollziehbaren und diskriminierungsfreien Betrieb der API-Webdienste. Sie konkretisieren die betrieblichen Mindestanforderungen an Verfügbarkeit, Performance, Resilienz, Überwachung, Störungsbehandlung und Sicherheit.

Konkrete Anforderungen, insbesondere zu Verfügbarkeit, Antwortzeiten, Timeouts, Retry-Logiken, Aufbewahrungsfristen und Eskalationsmechanismen, werden verbindlich in den jeweils zugehörigen Prozessbeschreibungen oder API-Spezifikationen festgelegt.

Sofern keine abweichenden Festlegungen getroffen werden, gelten die in dieser Guideline beschriebenen Grundsätze als Mindeststandard. Sie ersetzen keine vertraglich oder regulatorisch festgelegten Service-Level-Vereinbarungen, sondern sind unabhängig davon und ergänzend zu erfüllen.

3.1 Allgemeine Betriebsgrundsätze

MUST API-Webdienste müssen sicher, stabil, diskriminierungsfrei und nachvollziehbar betrieben werden.

MUST Der Betrieb muss so ausgestaltet sein, dass berechtigte Marktteilnehmer die vorgesehenen Funktionen und Prozesse ordnungsgemäß nutzen können.

MUST Einschränkungen des Betriebs sind auf das notwendige Maß zu begrenzen.

SHOULD Der Betrieb sollte so ausgestaltet sein, dass auch bei erhöhtem Anfrageaufkommen oder bei technischen Störungen eine geordnete Abwicklung der Marktkommunikation gewährleistet bleibt.

3.2 Verfügbarkeit und Wartung

MUST API-Webdienste müssen in dem für die jeweilige Marktkommunikation erforderlichen Umfang verfügbar sein.

MUST Geplante Wartungsarbeiten, Release-Wechsel und sonstige vorhersehbare Einschränkungen müssen rechtzeitig bekannt gemacht werden.

MUST Wartungen müssen so geplant und durchgeführt werden, dass Beeinträchtigungen der Marktkommunikation auf ein Minimum reduziert werden.

SHOULD Wartungsfenster sollten nach Möglichkeit außerhalb betriebsrelevanter Zeiten liegen.

3.3 Performance, Timeouts und Kapazität

MUST API-Webdienste müssen so dimensioniert sein, dass das zu erwartende Anfragevolumen sowie gleichzeitige Zugriffe berechtigter Kommunikationspartner ordnungsgemäß verarbeitet werden können.

MUST Antwortzeiten, Timeouts und sonstige leistungsbezogene Parameter müssen, soweit erforderlich, prozess- oder dienstspezifisch festgelegt werden.

MUST Technische Überlastsituationen müssen erkannt und geeignet behandelt werden.

SHOULD Maßnahmen zur Laststeuerung, Priorisierung oder kontrollierten Begrenzung sollten vorgesehen werden, soweit dies für einen stabilen Betrieb erforderlich ist.

3.4 Resilienz, Retry und Idempotenz

MUST API-Anbieter MÜSSEN grundsätzlich in der Lage sein, gleichzeitig allen berechtigten Clients (Kommunikationspartnern) einen Verbindungsaufbau zu ermöglichen.

MUST Der Betrieb der API-Webdienste muss so ausgestaltet sein, dass temporäre Störungen, Nichtverfügbarkeiten und technische Fehlerzustände beherrscht werden können.

MUST Die Retry-Regeln, insbesondere die maximale Anzahl von Wiederholungen, die Wartezeit zwischen Wiederholungen sowie gegebenenfalls Abbruchkriterien, MÜSSEN pro Prozess oder API-Webdienst verbindlich spezifiziert werden.

MUST Soweit für die sichere Wiederholung technischer Aufrufe erforderlich, müssen Mechanismen zur Sicherstellung der Idempotenz vorgesehen werden.

MUST NOT Unbegrenzte oder fachlich nicht definierte Wiederholungen sind nicht zulässig.

SHOULD Server SOLLTEN bei Überlastung, Ausfällen und anderen technischen Problemen geeignete HTTP-Statuscodes zurückgeben.

SHOULD Bei 429 Too Many Requests und 503 Service Unavailable SOLLTE der API-Anbieter, soweit möglich, den Header `Retry-After` setzen.

SHOULD Clients SOLLTEN Anfragen nach einem clientseitigen Timeout abbrechen und, soweit zulässig, gemäß den festgelegten Retry-Regeln erneut ausführen. [BDEW API 1.0a]

3.5 Monitoring, Logging und Nachvollziehbarkeit

MUST Für den Betrieb der API-Webdienste müssen geeignete Maßnahmen zur Überwachung und Protokollierung vorgesehen werden.

MUST Betriebsrelevante Ereignisse, Fehlerfälle und für die technische Nachvollziehbarkeit wesentliche Verarbeitungsschritte müssen in geeignetem Umfang erfasst werden.

MUST Die Protokollierung muss so ausgestaltet sein, dass Requests, Responses und technische Fehlerzustände im Bedarfsfall nachvollzogen und analysiert werden können.

SHOULD Monitoring und Logging sollten so ausgestaltet sein, dass Auffälligkeiten frühzeitig erkannt und Störungen zielgerichtet eingegrenzt werden können.

3.6 Störungen, Support und Betriebsstatus

MUST Für Störungen und sonstige betriebliche Beeinträchtigungen müssen geeignete Prozesse zur Erkennung, Bewertung, Eskalation und Behebung bestehen.

MUST Störungen und sonstige betriebliche Beeinträchtigungen, die Auswirkungen auf die technische Marktkommunikation haben, müssen gegenüber dem Data Hub angezeigt werden.

MUST NOT Eine bilaterale betriebliche Störungsmeldung an jeden einzelnen Marktteilnehmer ist für die in dieser Guideline geregelten Prozesse nicht vorgesehen.

MUST Der Data Hub muss einen zentralen Mechanismus zur Erfassung, Aktualisierung und Bereitstellung von Informationen über Wartungsfenster, Störungen und sonstige betriebliche Einschränkungen bereitstellen.

MUST Berechtigte Marktteilnehmer müssen den aktuellen Betriebsstatus betroffener Marktteilnehmer, Prozesse oder API-Webdienste in geeigneter Weise einsehen können.

MUST Der zentrale Statusmechanismus muss mindestens folgende Informationen enthalten:

- den betroffenen Marktteilnehmer oder Dienst,
- den betroffenen Prozess oder API-Webdienst,
- den aktuellen Betriebszustand,
- den Beginn der Einschränkung,
- soweit möglich das voraussichtliche Ende der Einschränkung,
- eine fachlich und technisch geeignete Kurzbeschreibung der Ursache oder Auswirkung,
- gegebenenfalls einen Verweis auf bestehende Wartungs- oder Störungsmeldungen.

MUST Für betriebliche Rückfragen und Störungsmeldungen müssen geeignete Kontakt- und Supportwege bereitgestellt werden.

SHOULD Die Störungsbearbeitung sollte so ausgestaltet sein, dass eine möglichst zeitnahe Wiederherstellung des ordnungsgemäßen Betriebs unterstützt wird.

3.7 Sicherheitsanforderungen im Betrieb

MUST Der Betrieb der API-Webdienste muss den Schutz der Verfügbarkeit, Integrität, Vertraulichkeit und Authentizität der Kommunikation gewährleisten.

MUST Hierzu sind angemessene technische und organisatorische Maßnahmen zum Schutz vor unberechtigtem Zugriff, Manipulation, missbräuchlicher Nutzung und Ausfall betriebsrelevanter Komponenten vorzusehen.

MUST Sicherheitsrelevante Ereignisse und Schwachstellen müssen unverzüglich bewertet, gemeldet und im erforderlichen Umfang behandelt werden.

SHOULD Sicherheitsmaßnahmen sollten regelmäßig überprüft und an den Stand der Technik sowie an die betriebliche Risikolage angepasst werden.

4 Fachliche Vorgaben

Die nachfolgenden Regelungen dienen der grundlegenden Konsistenz von API-Webdiensten. Die Konsistenz wird insbesondere durch einheitliche Regeln für URLs, Versionierung, Datenformate, Pflichtbestandteile, Rückmeldungsverhalten und die Struktur fachlicher Objekte sichergestellt. Die Schlüsselwörter „MUSS/MÜSSEN“ (MUST), „DARF/DÜRFEN NICHT“ (MUST NOT), „SOLL“ (SHALL), „SOLLTE“ (SHOULD) und „KANN/DARF“ (MAY) sind normativ zu verstehen.

4.1 Konsistenz

MUST Die API-Webdienste MÜSSEN nach einheitlichen und vorhersehbaren Gestaltungsprinzipien aufgebaut sein. Dies betrifft insbesondere die Benennung und Struktur von URLs, die konsistente Verwendung von HTTP-Methoden gemäß der jeweiligen API-Spezifikation sowie ein einheitliches technisches Verhalten bei Requests und Responses.

4.1.1 URL

URLs bilden die Grundlage von API-Webdiensten. Diese definieren den Kommunikationsendpunkt des API-Webdienstes.

MUST Eine URL darf keine Umlaute enthalten. [BDEW API 1.0a]

4.1.2 Struktur

SHOULD Nutzer können URLs einfach lesen und konstruieren

Beispiel einer gut strukturierten URL ist:

- https://api.teilnehmer.de/v1/allocations

Beispiel einer lesbaren und nicht strukturierten URL ist:

- `https://api.teilnehmer.de/33/55/zdfgd/rkfnhdrfeufuiefe-vcuberfiu5frf54/v1` [BDEW API 1.0a]

4.1.3 Regeln zum Aufbau

MUST Folgende Regeln sind beim Aufbau einer URL einzuhalten:

- URLs werden ohne abschließenden Schrägstrich gebildet.
- Die Pfadkomponente einer URL besteht ausschließlich aus Buchstaben, Zahlen, Unterstrichen, Bindestrichen sowie dem Schrägstrich als Segment-Trenner.
- Insbesondere werden keine weiteren Sonderzeichen verwendet.
- Die vollqualifizierte Versionsnummer wird nicht im URL-Pfad übertragen, sondern im HTTP-Header H2-API-Version. Im URL-Pfad wird ausschließlich die Major-Version mit dem Präfix v verwendet.

Zur besseren Lesbarkeit sollen Objekte / Ressourcen in `camelCase` Schreibweise benannt werden.

- Die Namen oder Bezeichner im URL-Pfad werden im Plural angegeben. [BDEW API 1.0a]
- Die Namen oder Bezeichner im URL-Pfad werden in englischer Sprache angegeben.
- Die Namen oder Bezeichner im URL-Pfad DÜRFEN keine Umlaute oder sonstige nicht-ASCII-Zeichen enthalten.

MAY Werte von Query-Parametern DÜRFEN nur solche Zeichen enthalten, die durch die jeweilige API-Spezifikation zugelassen und für die URI-Verwendung korrekt kodiert sind.

4.1.4 Länge

Die Länge einer URL wird durch diese Guideline nicht fest begrenzt. Technische Begrenzungen von Clients, Servern, Proxies oder Gateways sind bei der Spezifikation und Implementierung des jeweiligen API-Webdienstes zu berücksichtigen. Soweit URL-Längenbegrenzungen pro API-Webdienst erforderlich sind, MÜSSEN diese in der jeweiligen API-Spezifikation festgelegt werden.

4.1.5 Regeln zum Filtering

Folgende Regeln sind beim Filtering grundsätzlich einzuhalten:

MUST Filtering erfolgt ausschließlich bei `GET`-Operationen und ausschließlich über Query-Parameter.

Beispiel:

- `GET /allocations?networkCode=THE&calendarDay=2026-01-31`

MUST Filtering wird grundsätzlich nur auf Ressourcen-Sammlungen angewandt.

Beispiel zulässig:

- `GET /nominations?status=confirmed`

Beispiel unzulässig:

- `GET /nominations/12345?status=confirmed`

MUST Die Identifikation einer einzelnen Ressource erfolgt über den URL-Pfad und nicht über Filtering.

Beispiel zulässig:

- `GET /nominations/12345`

Beispiel unzulässig:

- `GET /nominations?id=12345`

MUST NOT JSON-Objekte dürfen im Rahmen des Filterings weder in Query-Parametern noch in HTTP-Headern übertragen werden.

Beispiel zulässig:

- `GET /allocations?networkCode=THE&calendarDay=2026-01-31`

Beispiele unzulässig:

- `GET /allocations?filter={"networkCode":"THE","calendarDay ":"2026-01-31"}`
- `GET /allocations mit Header H2-Filter: {"networkCode":"THE"}`

MUST Nicht unterstützte oder unzulässige Filterparameter führen zu `400 Bad Request`.

MUST Führt ein zulässiges Filtering zu keinem Treffer, wird `200 OK` mit einer leeren Ergebnisliste zurückgegeben.

SHOULD Filterparameter sollen sprechend, fachlich eindeutig und in englischer Sprache benannt werden.

MUST Mehrere Filterparameter werden grundsätzlich als UND-Verknüpfung interpretiert.

SHOULD Mehrere Werte desselben Filterparameters sollen durch wiederholte Angabe desselben Query-Parameters übergeben werden.

Der Abruf einzelner Ressourcen über eine Ressourcennummer im URL-Pfad (z. B. GET /nominations/{id}) ist nur zulässig, wenn diese Identifikationsnummer im jeweiligen fachlichen Prozess eindeutig definiert, marktweit eindeutig bekannt und konsistent verwendet wird.
Sofern keine solche prozessübergreifend eindeutige Identifikation existiert, sind Ressourcen ausschließlich über fachlich definierte Filterkriterien abzurufen

4.2 Versionierung und Änderungsmanagement

MUST Die Versionierung der Web-APIs Schnittstellen folgt der Semantik von Semantic Versioning 2.0:

<MAJOR>.<MINOR>.<PATCH>

- Die <MAJOR>-Version wird erhöht, wenn das API eine inkompatible Änderung beinhaltet.
- Die <MINOR>-Version wird erhöht, wenn neue Funktionalitäten, die kompatibel zur bisherigen API sind, veröffentlicht werden.
- Die <PATCH>-Version wird erhöht, wenn die Änderungen ausschließlich API-kompatible Bugfixes umfassen. Diese Änderungen sind zum Beispiel redaktionelle Änderungen/Hinweise/Änderung der Referenzen.

MUST Zur Differenzierung inkompatibler Schnittstellenversionen enthalten alle URL den MAJOR-Anteil mit einem kleiner „v“ als Präfix in der URL, wie im folgenden Beispiel gezeigt.

Beispiel:

- `https://api.teilnehmer.de/v1/allocations/`

MUST Die Antwort muss im HTTP-Header `H2-API-Version` die vollqualifizierte Versionsnummer (<MAJOR>.<MINOR>.<PATCH>, bspw. 3.1.0) beinhalten. [BDEW API 1.0a]

4.2.1 Änderungsmanagement

Während des Markthochlaufs erfolgt das Änderungsmanagement der API-Webdienste ausschließlich bei festgestelltem Änderungsbedarf. Ein turnusmäßiger Release-Zyklus ist nicht vorgesehen. Inkompatible Änderungen sind nur zulässig, soweit sie aus regulatorischen, fachlichen oder technischen Gründen erforderlich sind. Sie führen zu einer neuen Major-Version und sind vor ihrer Anwendung verbindlich bekannt zu machen. Fehlerbehebungen dürfen ausschließlich kompatible Änderungen enthalten, insbesondere ist die Einführung neuer Pflichtfelder unzulässig.

4.2.2 Abwärtskompatibilität

Eine Änderung an einem API-Webdienst ist abwärtskompatibel, wenn bestehende API-Nutzer der gleichen Major-Version den API-Webdienst weiterhin ohne Anpassung ihrer Implementierung verwenden können.

MUST Breaking Changes DÜRFEN ausschließlich mit einer neuen Major-Version eingeführt werden.
SHOULD Abwärtskompatible Änderungen SOLLEN innerhalb derselben Major-Version erfolgen.

MUST Abwärtskompatible Änderungen DÜRFEN bestehende Pflichtangaben, Datentypen, Wertebereiche, HTTP-Methoden, URL-Strukturen, Header, Query-Parameter, Statuscodes und fachliche Bedeutungen NICHT inkompatibel verändern.

MUST Die Einführung neuer Pflichtfelder, neuer verpflichtender Header oder neuer verpflichtender Query-Parameter ist innerhalb einer bestehenden Major-Version NICHT zulässig.

MAY Neue optionale Eigenschaften, optionale Header oder optionale Query-Parameter KÖNNEN innerhalb einer bestehenden Major-Version eingeführt werden, sofern bestehende API-Nutzer diese nicht übermitteln müssen und bestehende fachliche Verarbeitungen dadurch nicht verändert werden.

MUST Neue optionale Eigenschaften in Response-Objekten DÜRFEN innerhalb einer bestehenden Major-Version nur eingeführt werden, wenn bestehende API-Nutzer unbekannte Response-Eigenschaften ignorieren können oder die jeweilige API-Spezifikation die Erweiterbarkeit des Response-Objekts ausdrücklich vorsieht.

MUST API-Anbieter MÜSSEN bei abwärtskompatiblen Änderungen sicherstellen, dass API-Nutzer, die die bisherige Spezifikation derselben Major-Version verwenden, den API-Webdienst weiterhin ordnungsgemäß nutzen können.

MUST Eine abwärtskompatible Minor- oder Patch-Änderung DARF NICHT dazu führen, dass alle API-Nutzer verpflichtend auf die neue Minor- oder Patch-Version umstellen müssen.

SHOULD API-Nutzer SOLLTEN abwärtskompatible Minor- und Patch-Versionen nach Veröffentlichung übernehmen, soweit dies aus fachlichen, technischen oder betrieblichen Gründen zweckmäßig ist.

MUST Eine verpflichtende Umstellung aller API-Nutzer ist grundsätzlich nur bei inkompatiblen Änderungen zulässig. Inkompatible Änderungen MÜSSEN durch Erhöhung der Major-Version abgebildet und im Rahmen des Änderungsmanagements bekannt gemacht werden.

MUST Für inkompatible Änderungen MUSS ein verbindlicher Umstellungszeitraum festgelegt werden. Der Umstellungszeitraum MUSS in der jeweiligen API-Spezifikation oder Bekanntmachung angegeben werden und beträgt mindestens drei Monate ab Veröffentlichung der neuen Major-Version, sofern keine regulatorischen oder zwingenden technischen Gründe eine abweichende Frist erforderlich machen.

MUST Während des festgelegten Umstellungszeitraums MÜSSEN die bisherige, nicht abgekündigte Major-Version und die neue Major-Version parallel nutzbar sein, sofern dies technisch und fachlich möglich ist.

MUST Die Abkündigung einer API-Version MUSS eindeutig gekennzeichnet werden. Hierzu sind mindestens die folgenden Angaben zu verwenden:

- `deprecated: true`
- `deprecatedSince:` Zeitpunkt, ab dem die API-Version als abgekündigt gilt
- `sunsetAt`: Zeitpunkt, ab dem die API-Version nicht mehr genutzt werden darf und vom API-Anbieter abgeschaltet werden kann.

MUST Der Zeitpunkt `deprecatedSince` MUSS vor dem Zeitpunkt `sunsetAt` liegen.

MUST Das Datum `sunsetAt` MUSS den verbindlichen letzten Nutzungszeitpunkt der abgekündigten API-Version beschreiben.

MUST Eine abgekündigte API-Version DARF bis zum Zeitpunkt `sunsetAt` weiterhin genutzt werden, sofern in der Bekanntmachung keine abweichende Regelung festgelegt ist.

MUST Ab dem Zeitpunkt `sunsetAt` DARF die abgekündigte API-Version vom API-Anbieter abgeschaltet werden.

MUST Die Abkündigung MUSS in der API-Spezifikation und in der Bekanntmachung eindeutig beschrieben werden.

Die Angaben `deprecatedSince` und `sunsetAt` sind mindestens in der Bekanntmachung zu führen. Sofern sie in der OpenAPI-Spezifikation abgebildet werden, sind sie als H2-spezifische Erweiterungen zu modellieren.

Beispiel für Angaben in der Bekanntmachung:

- `deprecated: true`
- `deprecatedSince: 2026-10-01T00:00:00Z`
- `sunsetAt: 2027-01-01T00:00:00Z`

Beschreibung:

- Deprecated since 2026-10-01T00:00:00Z. Sunset at 2027-01-01T00:00:00Z.

4.2.3 Aufwärtskompatibilität

Ein API-Webdienst muss keine Aufwärtskompatibilität unterstützen. Parameter, Attribute oder sonstige Inhalte, die erst in einer neueren Schnittstellenspezifikation definiert sind, müssen von einer Umsetzung auf Basis einer älteren Schnittstellenspezifikation nicht verarbeitet werden.

4.3 Datentypen, JSON-Standardisierung und Schema-Grundregeln

4.3.1 Grundsatz

Die in API-Webdiensten verwendeten Datenstrukturen MÜSSEN eindeutig, interoperabel und maschinenlesbar beschrieben werden. Grundlage hierfür sind OpenAPI-Spezifikationen und JSON-Schemata.

MUST API-Spezifikationen MÜSSEN im Format OpenAPI 3.1 oder höher erstellt werden.

MUST JSON-Schemata MÜSSEN auf dem in OpenAPI 3.1 verwendeten JSON-Schema-Modell beruhen.

MUST Für den fachlichen Datenaustausch MÜSSEN JSON-Nachrichten gemäß [RFC8259] verwendet werden.

MUST JSON-Nachrichten MÜSSEN als UTF-8 ohne Byte Order Mark (BOM) kodiert werden.

MUST JSON-Nachrichten MÜSSEN die Anforderungen des I-JSON-Formats gemäß [RFC7493] einhalten.

MUST Die in der jeweiligen API-Spezifikation definierten Datentypen, Formate, Pflichtangaben, Wertebereiche, Enumerationen und Strukturvorgaben MÜSSEN eingehalten werden.

MUST JSON-Objekte DÜRFEN ausschließlich im HTTP-Body übertragen werden.

MUST NOT JSON-Objekte DÜRFEN NICHT in HTTP-Headern oder Query-Parametern übertragen werden.

Die Bildung eines internen JSON-Objekts ausschließlich für die Signaturberechnung und Signaturprüfung stellt keine Übertragung eines JSON-Objekts in HTTP-Headern oder Query-Parametern dar.

4.3.2 Benennung von Eigenschaften und Parametern

MUST Namen von JSON-Eigenschaften, Headern, Query-Parametern, Pfadparametern und Schema-Komponenten MÜSSEN in englischer Sprache gebildet werden.

MUST Namen von JSON-Eigenschaften, Headern, Query-Parametern, Pfadparametern und Schema-Komponenten DÜRFEN keine Umlaute enthalten.

MUST Namen von JSON-Eigenschaften und Schema-Komponenten DÜRFEN ausschließlich Buchstaben, Zahlen und die jeweils in der API-Spezifikation zugelassenen Trennzeichen verwenden.

MUST Die fachliche Bedeutung einer Eigenschaft MUSS in der jeweiligen API-Spezifikation eindeutig beschrieben werden.

MUST Eine Eigenschaft DARF innerhalb einer API-Spezifikation NICHT mehrfach mit unterschiedlicher fachlicher Bedeutung verwendet werden.

SHOULD JSON-Eigenschaften SOLLTEN in camelCase geschrieben werden.

SHOULD Schema-Komponenten SOLLTEN fachlich sprechend, eindeutig und wiederverwendbar benannt werden.

4.3.3 Primitive JSON-Datentypen

MUST Für jede fachliche Eigenschaft MUSS in der jeweiligen API-Spezifikation eindeutig festgelegt werden, welcher JSON-Datentyp zulässig ist.

MUST Zulässige primitive JSON-Datentypen sind:

JSON Data Type	Verwendung
string	Zeichenketten, Codes, IDs, Zeitangaben, Dezimalwerte mit exakter Darstellung
integer	Ganzzahlige Werte
number	Numerische Werte mit Nachkommastellen, sofern keine exakte Dezimaldarstellung erforderlich ist
boolean	Wahrheitswerte `true` oder `false`
array	Listen, Sequenzen oder geordnete Mengen
object	Strukturierte fachliche Objekte
null	Nur zulässig, wenn im Schema ausdrücklich erlaubt

MUST Fehlt die ausdrückliche Zulassung des Typs `null`, DARF eine Eigenschaft nicht mit dem Wert null übertragen werden.

MUST Numerische Werte MÜSSEN im jeweils spezifizierten Wertebereich liegen.

MUST Ganzzahlige Werte MÜSSEN als `integer` modelliert werden.

MUST Werte mit Nachkommastellen MÜSSEN als `number` oder, sofern exakte Dezimaldarstellung erforderlich ist, als string mit entsprechendem Pattern modelliert werden.

SHOULD Geldbeträge, Energiemengen, Messwerte und sonstige Werte, bei denen Rundungs- oder Präzisionsverluste fachlich relevant sein können, SOLLTEN nicht als `float` oder `double` modelliert werden, sondern als `decimalString` mit festgelegtem Format.

Beispiel `decimalString`:

- `type: string`
- `pattern: '^-?[0-9]+(\.[0-9]+)?$'`
- `example: '12345.678'`

4.3.4 Zugelassene Formate

Die nachfolgenden allgemeinen OpenAPI-/JSON-Schema-Formate DÜRFEN nur verwendet werden, wenn die jeweilige API-Spezifikation ihre Verwendung zulässt.

JSON Data Type	Value	Spezifikation	Beispiel
integer	int32	4 Byte vorzeichenbehaftete Integer-Nummer zwischen -2^{31} und $2^{31}-1$	`77210704`
integer	int64	8 Byte vorzeichenbehaftete Integer-Nummer zwischen -2^{63} und $2^{63}-1$	`772107100456824`
number	decimal	Vorzeichenbehaftete Dezimalnummer unbegrenzter Länge	`3.1415926535897932384626 43383279`
string	decimal-String	Vorzeichenbehaftete Dezimalnummer unbegrenzter Länge	„`3.14159265358979323846 2643383279`“
string	date	RFC 3339 Internet Profil – Subset von ISO 8601	“`2019-07-30`”
string	date-time	RFC 3339 Internet Profil – Subset von ISO 8601	“`2019-07-30T06:43:40.252Z`”
string	time	RFC 3339 Internet Profil – Subset von ISO 8601	“`06:43:40.252Z`”
string	duration	RFC 3339 Internet Profil – Subset von ISO 8601	“`P1DT30H4S`”
string	period	RFC 3339 Internet Profil – Subset von ISO 8601	“`2019-07-30T06:43:40.252Z/PT3H`”
string	password		“`secret`”
string	email	RFC 5322	“`example@exemple.de`”
string	idn-email	RFC 6531	“`hello@buecher.example`”
string	hostname	RFC 1034	“`www.test.de`”
string	idn-hostname	RFC 5890	“`buecher.example`”

JSON Data Type	Value	Spezifikation	Beispiel
string	ipv4	RFC 2673	“104.75.173.179”
string	ipv6	RFC 4291	“2600:1401:2::8a”
string	uri	RFC 3986	“https://www.test.de/“
string	uri-refe-rence	RFC 3986	“/clothing/”
string	uri-temp-late	RFC 6570	“/users/{id}”
string	iri	RFC 3987	“https://buecher.exa-mple/“
string	iri-refe-rence	RFC 3987	“/buecher-sport/”
string	uuid	RFC 9562	“018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21”
string	uuid-v7	RFC 9562	“018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21”
string	json-poin-ter	RFC 6901	“/items/0/id”
string	relative-json-poin-ter	Relative JSON Pointers	“1/id”
string	regex	Reguläre Ausdrücke wie in ECMA 262 beschrieben	“^[a-z0-9]+$”

4.3.5 Validierung und Wertebereiche

MUST Die in der API-Spezifikation angegebenen Formate MÜSSEN validiert werden.

MUST Formatangaben DÜRFEN NICHT ausschließlich dokumentierend verwendet werden, sofern sie in dieser Guideline oder in der jeweiligen API-Spezifikation als verbindlich festgelegt sind.

MUST Für jede fachliche Eigenschaft MUSS in der jeweiligen API-Spezifikation eindeutig festgelegt werden, ob sie verpflichtend, optional oder nullwertfähig ist.

MUST Pflichtangaben MÜSSEN im JSON-Schema über `required` definiert werden.

MUST Zulässige Wertebereiche MÜSSEN, soweit möglich, im JSON-Schema durch `minimum`, `maximum`, `minLength`, `maxLength`, `pattern`, `enum`, `minItems` oder `maxItems` beschrieben werden.

MUST Enumerationen MÜSSEN vollständig und eindeutig in der jeweiligen API-Spezifikation beschrieben werden.

MUST Nicht zugelassene Werte MÜSSEN zurückgewiesen werden.

MUST Zusätzliche, nicht im JSON-Schema definierte Eigenschaften MÜSSEN grundsätzlich zurückgewiesen werden, sofern die jeweilige API-Spezifikation keine ausdrückliche Erweiterbarkeit vorsieht.

SHOULD JSON-Objekte SOLLTEN grundsätzlich geschlossen modelliert werden.

4.3.6 Pflichtangaben, optionale Eigenschaften und Nullwerte in OpenAPI 3.1

In OpenAPI 3.1 werden Pflichtangaben und Nullwerte nach dem JSON-Schema-Modell beschrieben.

Das frühere OpenAPI-Attribut `nullable` wird in dieser Guideline NICHT verwendet. Nullwerte werden ausschließlich über den JSON-Schema-Typ `null` zugelassen.

MUST Eine verpflichtende Eigenschaft MUSS im JSON-Schema über `required` definiert werden.

MUST Eine Eigenschaft DARF nur dann mit dem Wert null übertragen werden, wenn der Typ null im jeweiligen Schema ausdrücklich zugelassen ist.

MUST Fehlt die ausdrückliche Zulassung des Typs `null`, DARF die Eigenschaft NICHT mit dem Wert `null` übertragen werden.

MUST Eine optionale Eigenschaft, die nicht befüllt werden soll, SOLLTE grundsätzlich weggelassen werden.

MUST NOT Der Wert `null` DARF NICHT verwendet werden, um eine nicht befüllte optionale Eigenschaft darzustellen, sofern null im Schema nicht ausdrücklich zugelassen ist.

MUST Das Fehlen einer Eigenschaft und die Übertragung einer Eigenschaft mit dem Wert `null` sind fachlich und technisch unterschiedlich zu behandeln.

Die nachfolgende Tabelle zeigt die zulässigen Modellierungsvarianten:

Modellierung	Eigenschaft darf fehlen	Wert null zulässig
Eigenschaft steht in required; Typ enthält nicht null	Nein	Nein
Eigenschaft steht in `required`; Typ enthält `null`	Nein	Ja
Eigenschaft steht nicht in required; Typ enthält nicht null	Ja	Nein

Eigenschaft steht nicht in required; Typ enthält null	Ja	Ja

Beispiel für eine verpflichtende, aber nullwertfähige Eigenschaft:

- `type: object`
- `required:`
  `- measuredValue`
- `properties:`
  `measuredValue:`
  `type:`
  `- number`
  `- "null"`

Beispiel für eine optionale, nicht nullwertfähige Eigenschaft:

- `type: object`
- `properties:`
- `comment:`
- `type: string`

4.3.7 Arrays, leere Arrays und Nullwerte in Arrays

MUST Eigenschaften vom Typ array MÜSSEN im JSON-Schema ausdrücklich als array modelliert werden.

MUST Für Arrays MÜSSEN die zulässigen Elemente über `items` beschrieben werden.

MUST Wenn ein Array verpflichtend ist, MUSS die Eigenschaft im JSON-Schema über `required` definiert werden.

MUST Eine verpflichtende Array-Eigenschaft DARF als leeres Array [] übertragen werden, sofern im JSON-Schema keine abweichende Mindestanzahl über `minItems` festgelegt ist.

MUST Soll ein Array mindestens ein Element enthalten, MUSS dies über `minItems`: 1 oder einen höheren Wert festgelegt werden.

MUST NOT Ein Array DARF KEINE Elemente mit dem Wert `null` enthalten, sofern `null` nicht ausdrücklich als zulässiger Elementtyp in `items` definiert ist.

MUST NOT Ein fehlendes Array, ein leeres Array [] und ein Array mit dem Wert `null` DÜRFEN NICHT gleichgesetzt werden.

Beispiel für ein verpflichtendes Array, bei dem ein leeres Array zulässig ist:

- `type: object`
- `required:`
  `- measuredValues`

- properties:
    measuredValues:
      type: array
      items:
        type: number

4.3.8 Geschlossene und erweiterbare JSON-Objekte

MUST In der jeweiligen API-Spezifikation MUSS festgelegt werden, ob ein JSON-Objekt zusätzliche, nicht ausdrücklich definierte Eigenschaften enthalten darf.

MUST Geschlossene JSON-Objekte MÜSSEN nicht definierte Eigenschaften zurückweisen.

Für geschlossene Objekte ist in JSON Schema `additionalProperties: false` zu verwenden.

Beispiel für ein geschlossenes Objekt:

- type: object
- additionalProperties: false
- required:
  - marketPartnerId
- properties:
    marketPartnerId:
      type: string
      pattern: '^[0-9]{13}$'

MAY Erweiterbare Objekte KÖNNEN zugelassen werden, wenn dies aus Gründen der Abwärtskompatibilität, Migration oder fachlichen Erweiterbarkeit erforderlich ist.

MUST Wenn zusätzliche Eigenschaften zugelassen werden, MUSS die jeweilige API-Spezifikation eindeutig beschreiben, ob und wie diese verarbeitet werden.

MUST Zusätzliche Eigenschaften DÜRFEN KEINE fachliche Bedeutung haben, sofern diese fachliche Bedeutung nicht in der jeweiligen API-Spezifikation beschrieben ist.

Beispiel für ein ausdrücklich erweiterbares Objekt:

```
type: object
required:
   - marketPartnerId
properties:
   marketPartnerId:
      type: string
      pattern: '^[0-9]{13}$'
additionalProperties: true
```

4.3.9 Abgrenzung zur Signaturberechnung

Die in diesem Kapitel beschriebenen Datentypen und JSON-Schemata betreffen die fachlich über HTTP übertragenen Daten, insbesondere den JSON-Body eines API-Aufrufs.

MUST Die fachlichen JSON-Schemata DÜRFEN NICHT um HTTP-Header, Query-Parameter oder Signaturinformationen erweitert werden, nur um die Signaturberechnung zu ermöglichen.

MUST Die Zusammenführung von Headern, Query-Parametern, Pfadinformationen und Payload in ein internes JSON-Objekt DARF ausschließlich für die Signaturberechnung und Signaturprüfung erfolgen.

MUST Das interne Signaturobjekt ist kein fachliches Nachrichtenformat und wird nicht als HTTP-Body übertragen.

Die Einzelheiten der signaturrelevanten JSON-Repräsentation werden in Kapitel 4.4 und in den Regelungen zur Inhaltsdatensicherungsebene beschrieben.

4.4 JSON-Schemata, Validierung und Feldreihenfolge

4.4.1 Verwendung von JSON-Schemata

MUST Für den Datenaustausch über API-Webdienste MÜSSEN JSON-Schemata verbindlich verwendet werden. Die JSON-Schemata dienen der eindeutigen formalen Beschreibung von Nachrichtenstrukturen, Datentypen, Pflichtfeldern, Optionalfeldern sowie zulässigen Wertebereichen der übertragenen Daten.

MUST Alle über den Übertragungsweg ausgetauschten JSON-Nachrichten MÜSSEN vollständig und ausnahmslos gegen das jeweils gültige JSON-Schema validierbar sein.

MUST NOT Nachrichten, die nicht schemakonform sind, DÜRFEN NICHT verarbeitet werden und SIND zurückzuweisen.

RECOMMENDED JSON-Nachrichten SOLLTEN bereits vor dem Versand durch den Sender gegen das jeweils gültige JSON-Schema validiert werden.

Änderungen an bestehenden JSON-Schemata oder die Einführung neuer Schemata unterliegen der Versionsverwaltung und sind konsistent zur jeweiligen API-Version zu veröffentlichen.

4.4.2 Reihenfolge von JSON-Feldern

JSON-Objekte besitzen keine fachlich verbindliche Reihenfolge ihrer Eigenschaften. Die Reihenfolge von Objektfeldern ist daher weder Bestandteil der fachlichen Semantik noch Gegenstand der Schema-Validierung.

MUST API-Webdienste MÜSSEN JSON-Objekte unabhängig von der Reihenfolge der enthaltenen Eigenschaften verarbeiten.

MUST NOT Die Verarbeitung eines API-Aufrufs DARF NICHT davon abhängen, in welcher Reihenfolge Eigenschaften innerhalb eines JSON-Objekts übertragen werden.

MUST NOT JSON-Schemata DÜRFEN NICHT dazu verwendet werden, eine feste Reihenfolge von Objektfeldern fachlich vorzugeben.

MUST NOT Eine feste Reihenfolge von Objektfeldern DARF NICHT über den Umweg einer Array-Struktur erzwungen werden, sofern es sich fachlich nicht tatsächlich um eine Liste, Sequenz oder geordnete Menge handelt.

MAY JSON-Arrays KÖNNEN verwendet werden, wenn eine fachliche Mehrfachstruktur abzubilden ist. Die Reihenfolge von Array-Elementen ist nur dann fachlich relevant, wenn dies in der jeweiligen Prozess- oder API-Spezifikation ausdrücklich beschrieben ist.

4.4.3 Fehlerbehandlung

MUST Ist die JSON-Nachricht syntaktisch ungültig oder fehlt ein erforderlicher Content-Type, MUSS mit HTTP `400 Bad Request` bzw. `415 Unsupported Media Type` geantwortet werden.

MUST Ist die JSON-Nachricht syntaktisch gültig, formal korrekt und der Content-Type zulässig, verletzt jedoch das fachlich verbindliche JSON-Schema, MUSS mit HTTP `422 Unprocessable Content` geantwortet werden.

MUST Negative Responses mit Response-Body MÜSSEN dem in der jeweiligen API-Spezifikation definierten ErrorResponse-Schema entsprechen.

4.5 Bestandteile eines API-Webdienstes

4.5.1 Namenskonvention für HTTP-Header

MUST Für neu definierte, proprietäre HTTP-Header ist ein einheitlicher, fachlich sprechender Namespace zu verwenden.

MUST NOT Für neu definierte HTTP-Header darf kein X-Präfix verwendet werden. [RFC6648]

MUST Proprietäre HTTP-Header dieser Guideline werden mit dem Präfix H2- gebildet.

MUST Auf das Präfix folgt ein fachlich sprechender Headername in englischer Sprache; mehrteilige Bezeichnungen werden durch Bindestriche getrennt.

SHOULD Die Header werden in der Spezifikation in einer einheitlichen kanonischen Schreibweise angegeben. Technisch ist die Groß- und Kleinschreibung von HTTP-Feldnamen nicht maßgeblich.

4.5.2 Verbindlich zu verwendende HTTP-Header

Die nachfolgend definierten HTTP-Header dienen der technischen Adressierung, Steuerung und Nachvollziehbarkeit der Kommunikation; eine fachliche oder prozessuale Verarbeitung hat ausschließlich auf Basis der hierfür vorgesehenen Nutzdaten und Prozessparameter zu erfolgen.

MUST Für Requests, Retry-Requests, synchrone Responses und asynchrone fachliche Responses gelten die folgenden verbindlichen Header:

Kontext	Verbindliche Header
Jeder Request	`H2-Transaction-Id, H2-Message-Sender, H2-Message-Receiver, H2-Business-Process`
Retry-Request	zusätzlich `H2-Initial-Transaction-Id`
Synchrone Response	`H2-API-Version; H2-Reference-Id`, sofern eine eindeutige Bezugnahme auf eine Anfrage erforderlich ist
Asynchrone fachliche Response	`H2-Transaction-Id, H2-Reference-Id, H2-Message-Sender, H2-Message-Receiver, H2-Business-Process, H2-API-Version`

MUST Falls eine Antwort eindeutig auf eine vorhergehende Anfrage Bezug nehmen muss, MUSS zusätzlich der Header H2-`Reference-Id` verwendet werden.

MUST Die Headerwerte MÜSSEN die folgenden Formate einhalten:

`H2-Message-Sender`

- OpenAPI Typ: string
- Zweck: Angabe der Marktpartner-ID des Absenders
- Zusätzliche Anforderung: Die Marktpartner-ID MUSS als 13-stellige numerische Zeichenfolge übertragen werden.
- Pattern: ^[0-9]{13}$
- Beispiel: 9871000123456

`H2-Message-Receiver`

- OpenAPI Typ: string
- Zweck: Angabe der Marktpartner-ID des Empfängers
- Zusätzliche Anforderung: Die Marktpartner-ID MUSS als 13-stellige numerische Zeichenfolge übertragen werden.
- Pattern: ^[0-9]{13}$
- Beispiel: 9871000123456

`H2-Transaction-Id`

- OpenAPI Typ: string

- OpenAPI Format: uuid
- Zweck: ID zur eindeutigen Identifikation eines Aufrufs
- Zusätzliche Anforderung: Es MUSS UUID Version 7 verwendet werden.
- Beispiel: 01aa2606-1180-7e42-85d9-8de28501f97d

`H2-Initial-Transaction-Id`

- OpenAPI Typ: string
- OpenAPI Format: uuid
- Zweck: Sicherstellung der Idempotenz
- Zusätzliche Anforderung: Es MUSS UUID Version 7 verwendet werden.
-
- Beispiel: 01aa2606-1180-7e42-85d9-8de28501f97d

`H2-Business-Process`

- OpenAPI Typ: string
- Zweck: Kennzeichnung des fachlichen Vorgangs bzw. Prozesses, dem ein API-Aufruf zuzuordnen ist. Der Header dient der eindeutigen technischen und fachlichen Einordnung des Aufrufs im Rahmen der jeweiligen Prozessspezifikation.
- Beispiel: nominationSubmission

`H2-API-Version`

- OpenAPI Typ: string
- Zweck: Angabe der vollqualifizierten API-Version der Antwort.
- Zusätzliche Anforderung: Die Version MUSS dem Format <MAJOR>.<MINOR>.<PATCH> entsprechen.
- Pattern: ^[0-9]+\.[0-9]+\.[0-9]+$
- Beispiel: 1.2.0

Beispiel eines vollständigen Header-Sets (ohne H2-API-Version der Antwort):

- `H2-Transaction-Id:` 018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21
- `H2-Message-Sender:` 9871000123456
- `H2-Message-Receiver:` 9871000654321
- `H2-Business-Process:` nominationSubmission

Beispiel für einen Retry:

- `H2-Transaction-Id:` 018f0d4f-12ab-7a90-9db3-1dc8e83a7123
- `H2-Initial-Transaction-Id:` 018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21
- `H2-Message-Sender:` 9871000123456
- `H2-Message-Receiver:` 9871000654321
- `H2-Business-Process:` nominationSubmission

MUST Falls sich eine Antwort auf eine Anfrage beziehen muss:

H2-`Reference-Id`

- OpenAPI Typ: string
- OpenAPI Format: uuid
- Zweck: Referenz auf die ursprüngliche Anfrage. Bei einem Retry ist die H2-Initial-Transaction-Id der ursprünglichen Anfrage zu referenzieren; andernfalls die H2-Transaction-Id der ursprünglichen Anfrage.
- Zusätzliche Anforderung: Es MUSS UUID Version 7 verwendet werden
- Pattern:
  ^[0-9a-f]{8}-[0-9a-f]{4}-7[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
- Beispiel: 01aa2606-1180-7e42-85d9-8de28501f97d

MUST NOT Die genannten Metadaten DÜRFEN NICHT als Query-Parameter übertragen werden.

4.5.3 Nutzung der H2-Transaction-Id, H2-Initial-Transaction-Id und H2-Reference-Id

MUST Clients müssen bei jeder Anfrage und bei jedem Retry immer eine neue H2-`Transaction-Id` vergeben und diese mitsenden. Bei einem Retry muss die H2-`Initial-Transaction-Id` angegeben werden mit der H2-`Transaction-Id` des initialen Aufrufs, um in diesen Fällen der wiederholten Anfragen technisch die gleichen Aufrufe identifizierbar zu machen (Idempotenz).

MUST Ist eine Anfrage kein Retry, dann darf die `H2-Initial-Transaction-Id` nicht angegeben werden, d. h., dieser Parameter darf in der Anfrage nicht enthalten sein.

MUST In einer Nachricht, die eine asynchrone Antwort auf eine vorherige Anfrage ist, muss im Feld „H2-`Reference-Id`“ die „H2-`Transaction-Id`“ des ursprünglichen Aufrufs bzw. wenn vorhanden, die „H2-`Initial-Transaction-Id`“ zurückgegeben werden, um eine eineindeutige Zuordnung zur ursprünglichen Anfrage zu ermöglichen.

Abbildung 1: Beispielhafte Darstellung der Nutzung von H2-Transaction-Id und H2-Initial-Transaction-Id

4.6 HTTP-Methoden

MUST Die für einen API-Webdienst zulässigen HTTP-Methoden MÜSSEN in der jeweiligen API-Spezifikation eindeutig festgelegt werden.

MUST Wird eine bekannte Ressource mit einer nicht unterstützten HTTP-Methode aufgerufen, MUSS der API-Anbieter mit `405 Method Not Allowed` antworten und im HTTP-Header „`Allow`“ die für diese Ressource zulässigen Methoden angeben.

MUST Die Verwendung von HTTP-Methoden MUSS konsistent entsprechend ihrer fachlichen und technischen Semantik erfolgen.

4.6.1 GET

MUST GET DARF ausschließlich für das lesende Abrufen von Ressourcen oder Informationen verwendet werden.

MUST NOT GET DARF NICHT für fachliche Zustandsänderungen, Prozessauslösungen oder Schreiboperationen verwendet werden.

MUST GET-Requests DÜRFEN keinen Request-Body enthalten.

SHOULD GET SOLLTE nur für Operationen verwendet werden, die sicher und ohne Seiteneffekte ausführbar sind.

Beispiel:

- `GET /allocations?networkCode=THE&calendarDay=2026-01-31`
- Abruf von Allokationsinformationen ohne fachliche Zustandsänderung.

4.6.2 POST

MUST POST MUSS verwendet werden, wenn ein fachlicher Vorgang ausgelöst, eine Nachricht eingereicht oder eine neue Ressource erzeugt wird, soweit die jeweilige Prozessspezifikation keine idempotente Erzeugung mittels PUT vorsieht.

SHOULD POST SOLLTE insbesondere für prozessuale Aufrufe mit asynchroner fachlicher Weiterverarbeitung verwendet werden.

MAY Eine erfolgreiche technische Entgegennahme KANN mit `202 Accepted` beantwortet werden, sofern die fachliche Rückmeldung asynchron erfolgt.

Beispiel:

- `POST /nominations`
- Einreichung einer Nominierung als fachlicher Vorgang.

4.6.3 PUT

MUST PUT DARF nur verwendet werden, wenn eine Ressource vollständig erzeugt oder vollständig ersetzt wird und die Operation idempotent ist.

MUST NOT PUT DARF NICHT verwendet werden, wenn lediglich ein fachlicher Vorgang ausgelöst oder eine Teiländerung vorgenommen werden soll.

Beispiel:

- `PUT /nominations/12345`
- vollständige Ersetzung der Nominierung 12345

4.6.4 PATCH

MUST PATCH DARF nur für partielle Änderungen an einer bestehenden Ressource verwendet werden.

MUST Die partiell änderbaren Attribute MÜSSEN in der jeweiligen API-Spezifikation ausdrücklich beschrieben werden.

MUST NOT PATCH DARF NICHT verwendet werden, wenn die fachliche Semantik einer vollständigen Ersetzung entspricht.

Beispiel:

- `PATCH /nominations/12345`
- partielle Änderung einzelner Attribute der Nominierung 12345

4.6.5 DELETE

MUST `DELETE` DARF nur verwendet werden, wenn die fachliche Prozessbeschreibung das Löschen einer Ressource ausdrücklich vorsieht.

MUST NOT `DELETE` DARF NICHT verwendet werden, wenn fachlich lediglich eine Stornierung, Abmeldung oder Statusänderung vorliegt, ohne dass die Ressource tatsächlich gelöscht wird.

Beispiel zulässig:

- `DELETE /nominations/12345`
- Löschen der Ressource Nominierung 12345, sofern die Prozessbeschreibung das Löschen ausdrücklich vorsieht.

Beispiel unzulässig:

- `DELETE /nominations/12345`
- unzulässig, wenn fachlich keine Löschung, sondern etwa eine Stornierung, Rücknahme oder Statusänderung vorgesehen ist.

Beispiel stattdessen:

- `POST /nominations/12345/cancellation`
- wenn fachlich eine Stornierung und keine Löschung der Ressource gemeint ist.

4.6.6 Weitere HTTP-Methoden

MUST NOT Andere HTTP-Methoden, insbesondere `TRACE`, `CONNECT` und `OPTIONS` als fachliche Prozessmethode, DÜRFEN NICHT für fachliche API-Webdienste verwendet werden, soweit ihre Nutzung nicht ausdrücklich technisch erforderlich und spezifiziert ist.

4.7 HTTP-Statuscodes

Jeder API-Aufruf wird unmittelbar mit einer synchronen HTTP-Response und einem HTTP-Statuscode beantwortet. Bei synchroner Verarbeitung können zusätzlich fachliche Nutzdaten im Response-Body zurückgegeben werden, insbesondere bei 200 OK. Erfolgt die fachliche Verarbeitung asynchron, ist 202 Accepted zu verwenden; in diesem Fall sollen keine fachlichen Nutzdaten im Response-Body enthalten sein. Technische Metadaten wie Statuscode und verpflichtende Response-Header bleiben davon unberührt. Weitere fachliche Rückmeldungen erfolgen nur, wenn dies in der jeweiligen Prozessbeschreibung vorgesehen ist. Die in der H2-Marktkommunikation über den Data Hub zu verwendenden HTTP-Statuscodes werden im folgenden Abschnitt verbindlich beschrieben.

4.7.1 Positive Response Codes

Code	Name	Erläuterung
200	OK	Die Anfrage wurde erfolgreich bearbeitet und das Ergebnis der Anfrage wird in der Antwort übertragen.
201	Created	Ressource wurde erfolgreich erzeugt
202	Accepted	Die Anfrage wurde zur Verarbeitung angenommen, aber die Verarbeitung ist noch nicht abgeschlossen.
204	No Content	Anfrage erfolgreich, keine Nutzdaten im Response-Body

4.7.2 Negative Response Codes

4.7.2.1 4xx – Fehler aufgrund ungültiger, unvollständiger oder unzulässiger Anfragen

Code	Name	Erläuterung
400	Bad Request	Die Anfrage ist syntaktisch ungültig oder enthält ungültige Parameter.
401	Unauthorized	Die Authentifizierung fehlt, ist ungültig oder konnte nicht erfolgreich geprüft werden.
403	Forbidden	Die Authentifizierung war erfolgreich, der authentifizierte Marktteilnehmer ist jedoch für den angefragten Prozess, die Ressource oder die Funktion nicht berechtigt.
404	Not Found	Die angeforderte Ressource konnte nicht gefunden werden
405	Method Not Allowed	Die Zielressource kann nicht aufgerufen werden, obwohl diese bekannt ist
408	Request Timeout	Der Client hat innerhalb der Zeit, die der Server zu warten bereit war, keine Anfrage gesendet.

409	Conflict	Konflikt mit bestehender Ressource
415	Unsupported Media Type	Medientyp in der Anfrage ist nicht unterstützt
422	Unprocessable Content	JSON Schema wurde verletzt
429	Too Many Requests	Client hat zu viele Anfragen in einem Zeitfenster gestellt (Ratenlimitierung). Clients sollten pausieren und zu einem späteren Zeitpunkt einen Retry versuchen.

4.7.2.2 5xx – Fehler aufgrund technischer Probleme bei der serverseitigen Verarbeitung

Code	Name	Erläuterung
500	Internal Server Error	Interner Fehler
501	Not Implemented	Der Server unterstützt die zur Erfüllung der Anfrage erforderlichen Funktionen nicht.
502	Bad Gateway	Der Server, der als Gateway fungierte, erhielt eine ungültige Antwort
503	Service Unavailable	Server kann temporär wegen Überlastung oder Wartungsarbeiten keine Anfragen bearbeiten. Clients sollten zu einem späteren Zeitpunkt einen Retry versuchen.
504	Gateway Timeout	Falls die Web-API als Proxy zu dahinterliegenden Systemen dient, kann bei deren Nichtverfügbarkeit dies angezeigt werden. Clients sollten zu einem späteren Zeitpunkt einen Retry versuchen.

4.8 Fachliche Objektmodellierung

MUST Fachliche Objekte MÜSSEN in der jeweiligen API-Spezifikation eindeutig beschrieben werden.

MUST Die jeweilige API-Spezifikation MUSS für jedes fachliche Objekt eindeutig festlegen:

- welche fachliche Bedeutung das Objekt hat,
- in welchem Prozesskontext das Objekt verwendet wird,
- welche Eigenschaften das Objekt enthält,
- welche Eigenschaften verpflichtend oder optional sind,
- welche Datentypen, Formate, Wertebereiche und Enumerationen zulässig sind,
- ob und in welchen Fällen der Wert null zulässig ist.

MUST Eine Eigenschaft DARF innerhalb einer API-Spezifikation NICHT mehrfach mit unterschiedlicher fachlicher Bedeutung verwendet werden.

MUST Eigenschaften mit gleicher fachlicher Bedeutung MÜSSEN über alle API-Webdienste hinweg konsistent verwendet werden.

MUST Die fachliche Bedeutung einer Eigenschaft MUSS in der jeweiligen API-Spezifikation eindeutig beschrieben werden.

MUST Wenn ein fachliches Objekt in mehreren API-Webdiensten verwendet wird, MUSS die fachliche Bedeutung des Objekts in allen Verwendungen identisch sein.

MUST Abweichungen von wiederverwendbaren Referenzschemata MÜSSEN in der jeweiligen API-Spezifikation ausdrücklich begründet und beschrieben werden.

SHOULD Wiederverwendbare fachliche Objekte SOLLTEN unter components/schemas definiert und über Referenzen eingebunden werden.

SHOULD Wiederverwendbare Basistypen, insbesondere Marktpartner-ID, Zeitintervall, Messwert, Einheit, Statuscode und Rollenangaben, SOLLTEN zentral als Referenzschemata gepflegt werden.

SHOULD Fachliche Objekte SOLLTEN so weit wie möglich prozessübergreifend wiederverwendbar modelliert werden, sofern dadurch keine fachliche Mehrdeutigkeit entsteht.

4.9 Bekanntmachung

Änderungen an API-Webdiensten, Schnittstellenspezifikationen sowie fachlichen und technischen Vorgaben sind in geeigneter Weise bekannt zu machen. Die Bekanntmachung muss mindestens den Gegenstand der Änderung, die betroffenen API-Webdienste oder Prozesse, die Einordnung als kompatible oder inkompatible Änderung, den verbindlichen Umsetzungszeitpunkt sowie gegebenenfalls Übergangs- und Migrationsfristen enthalten. Führt eine Änderung zu einer neuen Major-Version, ist dies ausdrücklich kenntlich zu machen. Fehlerbehebungen und sonstige kompatible Änderungen sind ebenfalls bekannt zu machen.

5 Quellen

[RFC2119] Bradner, S. *Key words for use in RFCs to Indicate Requirement Levels.* IETF RFC 2119. March 1997. https://www.rfc-editor.org/rfc/rfc2119

[RFC8259] Bray, T. *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF RFC 8259. December 2017. https://www.rfc-editor.org/rfc/rfc8259

[RFC7493] Bray, T. *The I-JSON Message Format.* IETF RFC 7493. March 2015. https://www.rfc-editor.org/rfc/rfc7493

[RFC9110] Fielding, R., Nottingham, M., and Reschke, J. *HTTP Semantics.* IETF RFC 9110. June 2022. https://www.rfc-editor.org/rfc/rfc9110

[RFC8174] Leiba, B. *Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words.* IETF RFC 8174. May 2017. https://www.rfc-editor.org/rfc/rfc8174

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L. *Uniform Resource Identifier (URI): Generic Syntax.* IETF RFC 3986. January 2005. https://www.rfc-editor.org/rfc/rfc3986

[RFC6648] Saint-Andre, P., Crocker, D., and Nottingham, M. *Deprecating the "X-" Prefix and Similar Constructs in Application Protocols.* IETF RFC 6648. June 2012. https://www.rfc-editor.org/rfc/rfc6648

[RFC8785] Rundgren, A., Jordan, B., and Erdtman, S. *JSON Canonicalization Scheme (JCS).* IETF RFC 8785. June 2020. https://www.rfc-editor.org/rfc/rfc8785

[RFC9562] Davis, K., Peabody, B., and Eubanks, C. Universally *Unique IDentifiers (UUIDs).* IETF RFC 9562. May 2024. https://www.rfc-editor.org/rfc/rfc9562

[OpenAPI 3.1] OpenAPI Initiative. *OpenAPI Specification Version 3.1.* https://spec.openapis.org/oas/v3.1.0

[JSON Schema 2020-12] JSON Schema Organization. *JSON Schema Draft 2020-12 Core and Validation Specifications.* https://json-schema.org/specification

[SemVer 2.0.0] Preston-Werner, T. *Semantic Versioning 2.0.0.* https://semver.org

[BNetzA BK7-24-01-014] Bundesnetzagentur. *Beschluss im Festlegungsverfahren WasABi in Sachen Wasserstoff Ausgleichs- und Bilanzierungsgrundmodell.*

[BNetzA BK7-24-01-015] Bundesnetzagentur. *Beschluss im Festlegungsverfahren WaKandA in Sachen Wasserstoff Kapazitäten-Grundmodell und Abwicklung des Netzzugangs.*

[BDEW API 1.0a] BDEW. *API-Guideline. Version 1.0a.* October 2024. https://bdew-mako.de/pdf/API_Guideline_1_0a_20241001.pdf