API Guideline Wasserstoffmarkt V1.0

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface / Anwendungsschnittstelle
BDEW	Bundesverband der Energie- und Wasserwirtschaft
BNetzA	Bundesnetzagentur
BOM	Byte Order Mark
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certificate Authority
CRL	Certificate Revocation List
CP	Certificate Policy
DNS	Domain Name System
ECMA	European Computer Manufacturers Association / Organisation für die Standardisierung von JavaScript
EDI	Electronic Data Interchange
EDM	Energiedatenmanagement (EDM-System)
EMT	Externer Marktteilnehmer (Rolle innerhalb der SM-PKI)
etc	Et cetera / und so weiter
ggf	gegebenenfalls
H2-	Namespace-Präfix der in dieser Guideline definierten proprietären HTTP-Header
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I-JSON	Internet JSON (restriktives JSON-Profil gemäß RFC 7493)
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JWS	JSON Web Signature
LDAP	Lightweight Directory Access Protocol
MP-ID	Marktpartner-ID
MsbG	Messstellenbetriebsgesetz
mTLS	Mutual Transport Layer Security
OpenAPI	Open Application Programming Interface Specification
OU	Organizational Unit (Feld im Zertifikat)
PROD	Produktivumgebung
RFC	Request for Comments / technische Dokumente und Standards der Internet Engineering Taskforce
SemVer	Semantic Versioning
SM-PKI	Smart Meter Public Key Infrastructure
Sub-CA	Subordinate Certificate Authority

TEST	Test- und Abnahmeumgebung
TLS	Transport Layer Security
TR-03116-3	Technische Richtlinie des BSI für kryptographische Verfahren (Teil 3: Intelligente Messsysteme)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
WaKandA	Wasserstoff-Kapazitäten-Grundmodell und Abwicklung des Netzzugangs
WasABi	Wasserstoff-Ausgleichs- und Bilanzierungsgrundmodell
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.

Hinweis: Der Begriff „fachlich“ grenzt diese Ebene von rein technischen Transport- und Sicherheitsvorgaben ab, etwa TLS, mTLS, JWS, Zertifikatsprüfung oder Netzwerkadressierung. Mit "fachlichen Vorgaben" sind hier nicht die fachlichen oder prozessualen Marktregeln gemeint, sondern die fachliche Semantik und Modellierung der API-Schnittstelle, insbesondere Ressourcen, Datenobjekte, Operationen, Statuscodes und Validierungsregeln.

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 verbindliche 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 und über den Data Hub laufen,
- 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), „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	Betreiber des technischen API-Endpunkts einer konkreten Kommunikationsstrecke. Die Rolle ist für jede Kommunikationsstrecke gesondert zu bestimmen.
API-Nutzer	Technischer Client, der einen konkreten API-Endpunkt aufruft.
Kommunikationsendpunkt	URL und Port (URI) des API-Webdienstes.
Data Hub	Zentrale technische Kommunikations- und Datenaustauschplattform.
Marktpartner	Natürliche oder juristische Person, die in einer festgelegten Marktrolle an der Marktkommunikation teilnimmt. Für fachliche Teilnehmer der Marktkommunikation wird in dieser Guideline einheitlich der Begriff Marktpartner verwendet; der Begriff Teilnehmer bleibt im Zusammenhang mit der SM-PKI unberührt.

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. Kommunikation außerhalb der in WasABi und WaKandA geregelten Marktprozesse werden in der Guideline nicht erfasst (beispielsweise interne

Backend-zu-Backend-Kommunikation). 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 Authentisierung und Autorisierung sicher und ermöglicht eine feingranulare Steuerung der Zugriffe auf Daten und Funktionen. Dies setzt ein konsistentes Rollen- und Rechtemodell voraus. Die Authentisierung der Kommunikationspartner erfolgt zertifikatsbasiert; die Autorisierung erfolgt auf Grundlage des Rollen- und Rechtemodells.

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.

2.3.4. Kommunikations- und Vertrauensmodell

Im Sinne dieses Kapitels ist der technische Signierer das System, das eine Nachricht für die jeweils aktuelle Kommunikationsstrecke erzeugt und mit seinem `EMT.API`-Zertifikat signiert. Der technische Signierer kann vom ursprünglichen fachlichen Absender abweichen.

MUST Der Data Hub MUSS vor einer fachlichen Verarbeitung mindestens:

- die mTLS-Authentisierung prüfen,
- die Zertifikatsgültigkeit und den Sperrstatus prüfen,
- die Berechtigung des technischen Signierers prüfen,
- die Signatur der Nachricht prüfen und
- bei einem initialen Aufruf eines Marktpartners prüfen, dass `H2-Message-Sender` der dem mTLS-Client-Zertifikat zugeordneten Marktpartner-ID entspricht.

MUST Für jede Kommunikationsstrecke MUSS zwischen dem ursprünglichen fachlichen Absender und dem technischen Signierer der aktuellen Kommunikationsstrecke unterschieden werden.

MUST Ist `H2-Forwarded-By` nicht vorhanden, MUSS `H2-Message-Sender` den technischen Signierer bezeichnen. Die MP-ID des Signaturzertifikats und die MP-ID des mTLS-Client-Zertifikats MÜSSEN `H2-Message-Sender` entsprechen.

MUST Ist `H2-Forwarded-By` vorhanden, MUSS `H2-Forwarded-By` den technischen Signierer der aktuellen Kommunikationsstrecke bezeichnen. Die MP-ID des Signaturzertifikats und die MP-ID des mTLS-Client-Zertifikats MÜSSEN `H2-Forwarded-By` entsprechen. `H2-Message-Sender` und `H2-Message-Sender-Role` MÜSSEN weiterhin den ursprünglichen fachlichen Absender bezeichnen.

MUST NOT Ein initialer Request eines Marktpartners DARF `H2-Forwarded-By` NICHT enthalten. Der Data Hub MUSS einen solchen Request vor der fachlichen Verarbeitung mit `400 Bad Request` ablehnen.

MUST Soweit der Data Hub zur weiteren Verarbeitung einen technischen Endpunkt eines Marktpartners aufruft, MUSS er eine neue mTLS-Verbindung aufbauen und einen neuen, durch den Data Hub signierten HTTP-Request erzeugen.

MUST Der ausgehende Request MUSS eine neue `H2-Transaction-Id` erhalten. `H2-Initial-Transaction-Id` DARF nur verwendet werden, wenn der Data Hub diesen ausgehenden Request selbst als Retry gemäß Teil 1, Kapitel 4.5.3 erneut überträgt; in diesem Fall MUSS sie die `H2-Transaction-Id` des ersten ausgehenden Requests derselben Kommunikationsstrecke referenzieren. Transaktionskennungen der eingehenden Kommunikationsstrecke DÜRFEN NICHT als `H2-Initial-Transaction-Id` der ausgehenden Kommunikationsstrecke übernommen werden.

MUST Der Data Hub DARF die Header `H2-Signature`, `H2-Digest` und `H2-Cert` der eingehenden Kommunikationsstrecke NICHT unverändert weiterleiten. Nach erfolgreicher Prüfung MUSS er die Inhaltsdatensicherung für die ausgehende Kommunikationsstrecke mit dem Signaturschlüssel des eigenen `SIG`-Zertifikats aus seinem gültigen `EMT.API`-Zertifikatstripel neu erzeugen.

MUST Der Data Hub MUSS Angaben zum ursprünglichen fachlichen Absender vor deren Übernahme und erneuter Signatur anhand der eingehenden mTLS-Verbindung und Nachrichtensignatur erfolgreich geprüft haben.

MUST `H2-Message-Receiver` und `H2-Message-Receiver-Role` MÜSSEN den Empfänger der jeweiligen Kommunikationsstrecke bezeichnen. Der fachliche Endempfänger eines Aufrufs ergibt sich aus der `providerId` im URL-Pfad.

MUST Ein durch einen Marktpartner initiierter API-Aufruf MUSS am Data Hub terminieren. Eine direkte technische Kommunikation zwischen Marktpartnern ist nicht vorgesehen. Die anschließende Weiterleitung an marktpartnerbezogene Endpunkte erfolgt ausschließlich durch den Data Hub.

MUST Empfängt der Data Hub aus einer nachgelagerten Kommunikationsstrecke eine signaturpflichtige synchrone Response, MUSS er deren Signatur prüfen. Für die Rückgabe an den ursprünglichen API-Nutzer MUSS der Data Hub eine neue Response der aktuellen Kommunikationsstrecke erzeugen und, soweit diese nach Teil 2, Kapitel 2.3 signaturpflichtig ist, mit seinem eigenen `EMT.API`-Zertifikat signieren. Die Signatur der nachgelagerten Kommunikationsstrecke DARF NICHT als Signatur der vorgelagerten Kommunikationsstrecke verwendet werden.

MUST Empfängt der Data Hub aus einer nachgelagerten Kommunikationsstrecke eine asynchrone fachliche Response, MUSS er deren mTLS-Authentisierung, Signatur, `H2-Reference-Id`, Berechtigung und fachliche Zuordnung prüfen.

MUST Der Data Hub MUSS die `H2-Reference-Id` der nachgelagerten Kommunikationsstrecke anhand seiner gespeicherten Transaktionszuordnung auf die effektive Transaktions-ID der vorgelagerten Kommunikationsstrecke abbilden.

MUST Für die Weiterleitung an den ursprünglichen Marktpartner MUSS der Data Hub einen neuen, durch ihn signierten HTTP-Request mit neuer `H2-Transaction-Id` erzeugen. Dessen `H2-Reference-Id` MUSS die effektive Transaktions-ID des ursprünglichen Aufrufs auf der vorgelagerten Kommunikationsstrecke enthalten.

MUST NOT Transaktionskennungen oder Signaturen der nachgelagerten Kommunikationsstrecke DÜRFEN unverändert weitergeleitet werden.

MUST Der Data Hub MUSS die Zuordnung zwischen vorgelagerter und nachgelagerter Kommunikationsstrecke mindestens bis zum Ende des fachlich zulässigen Antwortzeitraums aufbewahren.

MUST Für jede Weiterleitung MÜSSEN mindestens die ursprüngliche `H2-Transaction-Id`, das Ergebnis der Signaturprüfung, der Hashwert der fachlichen Nutzdaten, die MP-ID des technischen Signierers und die neu vergebene `H2-Transaction-Id` nachvollziehbar protokolliert werden.

Eine Validierung der fachlichen Nutzdaten gegen das verbindliche JSON-Schema erfolgt durch den Data Hub nicht; sie obliegt der fachlich verarbeitenden Stelle.

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 Marktpartner die vorgesehenen Funktionen und Prozesse ordnungsgemäß nutzen können.

MUST Einschränkungen des Betriebs MÜSSEN auf das notwendige Maß begrenzt werden.

MUST Der Betrieb MUSS 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 mit dem in den geltenden Betriebsregelungen festgelegten Vorlauf 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 nach den Vorgaben des Kapitels 3.4 kontrolliert 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 Die für einen API-Webdienst zulässige Anzahl gleichzeitiger Verbindungen und Requests MUSS anhand des erwarteten Marktvolumens dimensioniert und in der jeweiligen Prozessbeschreibung bzw. API-Spezifikation festgelegt werden.

MAY Der API-Anbieter DARF dokumentierte technische Obergrenzen durchsetzen.

MUST Bei einer vorübergehenden Überschreitung MÜSSEN `429 Too Many Requests` oder `503 Service Unavailable` verwendet werden.

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 DÜRFEN NICHT erfolgen.

SHOULD Bei `429 Too Many Requests` und `503 Service Unavailable` SOLLTE der API-Anbieter den Header `Retry-After` setzen, sofern ein sinnvoller Zeitpunkt oder Zeitraum für einen erneuten Aufruf bestimmt werden kann.

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

MUST Berechtigte Marktpartner MÜSSEN den aktuellen Betriebsstatus betroffener Marktpartner, Prozesse oder API-Webdienste über den zentralen Statusmechanismus des Data Hub abrufen können.

MUST Der zentrale Statusmechanismus MUSS mindestens folgende Informationen enthalten:

- den betroffenen Marktpartner 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 dokumentierte und erreichbare Kontakt- und Supportwege bereitgestellt werden.

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

Eine bilaterale betriebliche Störungsmeldung an einzelne Marktpartner ist für die in dieser Guideline geregelten Prozesse nicht erforderlich. Die Information der Marktpartner erfolgt über die in diesem Kapitel festgelegten zentralen Melde- und Statusmechanismen.

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 MÜSSEN angemessene technische und organisatorische Maßnahmen zum Schutz vor unberechtigtem Zugriff, Manipulation, missbräuchlicher Nutzung und Ausfall betriebsrelevanter Komponenten vorgesehen werden.

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

MUST Sicherheitsmaßnahmen MÜSSEN 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) „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 NOT Eine URL DARF KEINE Umlaute enthalten. [BDEW API 1.0a]

4.1.2. Struktur

MUST URLs MÜSSEN für Nutzer einfach lesbar und konstruierbar sein.

Beispiel einer gut strukturierten URL ist:

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

Beispiel einer nicht 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 MÜSSEN beim Aufbau einer URL eingehalten werden:

- 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. Sie wird ausschließlich vom Server in der Antwort im HTTP-Header `H2-API-Version` mitgeteilt. Im URL-Pfad wird ausschließlich die Major-Version mit dem Präfix v verwendet.

SHOULD Zur besseren Lesbarkeit SOLLTEN Objekte und 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 MUSS ausschließlich bei `GET`-Operationen und ausschließlich über Query-Parameter erfolgen.

Beispiel:

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

MUST Filtering MUSS grundsätzlich nur auf Ressourcen-Sammlungen angewandt werden.

Beispiel zulässig:

- `GET /nominations?status=confirmed`

Beispiel unzulässig:

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

SHOULD Die Identifikation einer einzelnen Ressource SOLLTE über den URL-Pfad und nicht über Filtering erfolgen.

Beispiel zulässig:

- `GET /nominations/{nominationId}`

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 MÜSSEN zu `400 Bad Request` führen.

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

MUST Filterparameter MÜSSEN sprechend, fachlich eindeutig und in englischer Sprache benannt werden.

MUST Mehrere Filterparameter MÜSSEN grundsätzlich als UND-Verknüpfung interpretiert werden.

SHOULD Mehrere Werte desselben Filterparameters SOLLTEN 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, berechtigt ist 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-API-Schnittstellen MUSS der Semantik von Semantic Versioning 2.0 folgen:

`<MAJOR>.<MINOR>.<PATCH>`

- Die `<MAJOR>`-Version wird erhöht, wenn die 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 MÜSSEN alle URLs den `MAJOR`-Anteil mit einem kleinen „v“ als Präfix in der URL enthalten, wie im folgenden Beispiel gezeigt.

Beispiel:

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

Die vorstehende Regel gilt für die Versionierung des jeweils aufgerufenen API-Webdienstes. Das Segment `<majorVersion>` in fachlichen API-Aufrufen an den Data Hub gemäß Teil 2, Kapitel 4.4 ist Bestandteil des Verzeichnisschlüssels und wird abweichend hiervon ohne das Präfix v übertragen.

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

MUST Clients MÜSSEN die gewünschte API-Version ausschließlich über die Major-Version im URL-Pfad auswählen. Eine Auswahl einer bestimmten Minor- oder Patch-Version durch den Client ist nicht vorgesehen; innerhalb einer Major-Version antwortet der Server mit der jeweils aktuell bereitgestellten Version.

Ein in einem Request enthaltener Header `H2-API-Version` wird durch den Server ignoriert und hat keine Auswirkung auf die Verarbeitung.

4.2.1. Änderungsmanagement

MUST Fehlerbehebungen DÜRFEN ausschließlich kompatible Änderungen enthalten.

MUST NOT Die Einführung neuer Pflichtfelder DARF im Rahmen von Fehlerbehebungen NICHT erfolgen.

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 SOLLTEN innerhalb derselben Major-Version erfolgen.

MUST NOT Abwärtskompatible Änderungen DÜRFEN bestehende Pflichtangaben, Datentypen, Wertebereiche, HTTP-Methoden, URL-Strukturen, Header, Query-Parameter, Statuscodes und fachliche Bedeutungen NICHT inkompatibel verändern.

MUST Neue Pflichtfelder, neue verpflichtende Header oder neue verpflichtende Query-Parameter DÜRFEN innerhalb einer bestehenden Major-Version NICHT eingeführt werden.

MAY Neue optionale Eigenschaften, optionale Header oder optionale Query-Parameter KÖNNEN innerhalb einer bestehenden Major-Version eingeführt werden, sofern bestehende API-Nutzer nicht zu deren Übermittlung verpflichtet werden 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 NOT Eine abwärtskompatible Minor- oder Patch-Änderung DARF NICHT dazu führen, dass alle API-Nutzer zu einer Umstellung auf die neue Minor- oder Patch-Version verpflichtet werden.

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 DARF grundsätzlich nur bei inkompatiblen Änderungen erfolgen. 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 MÜSSEN mindestens die folgenden Angaben verwendet werden:

- `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.

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

MAY 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.

MUST Die Angaben `deprecatedSince` und `sunsetAt` MÜSSEN mindestens in der Bekanntmachung geführt werden. Sofern sie in der OpenAPI-Spezifikation abgebildet werden, MÜSSEN sie als `H2-spezifische` Erweiterungen modelliert werden.

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

MAY Ein API-Webdienst KANN ohne Unterstützung der Aufwärtskompatibilität spezifiziert werden. 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.

4.2.4. Bekanntmachung

MUST 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, MUSS dies ausdrücklich kenntlich gemacht werden. Fehlerbehebungen und sonstige kompatible Änderungen MÜSSEN ebenfalls bekannt gemacht 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 in einer durch die API-Governance freigegebenen Version der OpenAPI Specification erstellt werden. Bis zu einer abweichenden Festlegung ist OpenAPI 3.1 zu verwenden.

MUST Die OpenAPI-Spezifikation der synchronen Verzeichnisdienst-Pflege-API MUSS die in Teil 2, Kapitel 4.7 festgelegte optimistische Parallelitätskontrolle mit `ETag`, `If-Match` sowie, soweit für die jeweilige Operation vorgesehen, `If-None-Match` und den Statuscodes `412 Precondition Failed` und `428 Precondition Required` (die Statuscodes 412 und 428 sind in Teil 1, Kapitel 4.7 katalogisiert) vollständig und maschinenlesbar abbilden. Darüber hinaus MÜSSEN die für den jeweiligen Request- und Response-Kontext verbindlichen H2-Signaturheader einschließlich ihrer Pflichtbedingungen beschrieben werden.

MUST Der Header `If-Match` MUSS als starker Entity-Tag mit dem Muster ^"rev-[0-9]+"$ beschrieben werden; der Response-Header `ETag` MUSS demselben Muster folgen. Der Header `If-None-Match` MUSS für die Anlageoperation den festen Wert * zulassen.

MUST `If-Match` beziehungsweise `If-None-Match` sind Bestandteil der Signaturrepräsentation gemäß Teil 2, Kapitel 2.3 und [JWS-SPEC], damit der übertragene Vorbedingungswert auf dem Übertragungsweg nicht unbemerkt verändert werden kann.

MUST Für einen signaturpflichtigen synchronen Request MÜSSEN in der OpenAPI-Spezifikation mindestens die Header `H2-Transaction-Id`, `H2-Message-Sender`, `H2-Message-Sender-Role`, `H2-Message-Receiver`, `H2-Message-Receiver-Role`, `H2-Business-Process`, `H2-Signature`, `H2-Digest` und `H2-Cert`, bei vorhandenem Body zusätzlich `Content-Type` sowie, soweit erforderlich, `If-Match` oder `If-None-Match` beschrieben werden.

MUST Für eine signaturpflichtige synchrone Response mit JSON-Body MÜSSEN mindestens die Header `H2-Transaction-Id`, `H2-Reference-Id`, `H2-API-Version`, `H2-Signature`, `H2-Digest`, `H2-Cert` und `Content-Type` sowie, soweit ein aktueller Verzeichniseintrag zurückgegeben wird, `ETag` beschrieben werden. Ein gesonderter Signaturzeit-Header ist nicht erforderlich, da der Signaturzeitpunkt aus dem UUID-Version-7-Zeitwert der aktuellen `H2-Transaction-Id` bestimmt wird (Teil 1, Kapitel 4.5.3).

MUST JSON-Schemata MÜSSEN auf dem von der freigegebenen OpenAPI-Version unterstützten 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 Requests und Responses mit JSON-Body MÜSSEN den HTTP-Header `Content-Type` mit dem Wert `application/json` verwenden. Die Angabe des Zeichensatzparameters `charset=utf-8` ist zulässig; abweichende Zeichensätze sind unzulässig.

MUST Requests mit JSON-Body ohne `Content-Type`-Header oder mit einem anderen Medientyp als `application/json` MÜSSEN mit HTTP `415 Unsupported Media Type` abgelehnt werden.

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

MUST Bei HTTP-basierten API-Webdiensten DÜRFEN JSON-Objekte 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 NOT Eine Eigenschaft DARF innerhalb einer API-Spezifikation NICHT mehrfach mit unterschiedlicher fachlicher Bedeutung verwendet werden.

SHOULD JSON-Eigenschaften SOLLTEN in `lowerCamelCase` geschrieben werden.

SHOULD Enum-Werte bzw. fachliche Werte SOLLTEN in `UpperCamelCase` 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 Es DÜRFEN ausschließlich die folgenden primitiven JSON-Datentypen verwendet werden:

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 NOT 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 Ganzzahlige JSON-Werte DÜRFEN ausschließlich im Bereich von `-9007199254740991` bis `9007199254740991` übertragen werden. Ganzzahlige fachliche Werte außerhalb dieses Bereichs MÜSSEN als Zeichenfolge mit einem in der jeweiligen API-Spezifikation festgelegten Pattern modelliert werden.

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

MUST Geldbeträge, Energiemengen, Messwerte und sonstige Dezimalwerte, bei denen Rundungs- oder Präzisionsverluste fachlich relevant sind, MÜSSEN als `decimalString` mit verbindlichem Format und Pattern modelliert werden.

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 $2^{63}-1$ Das Format darf nur verwendet werden, wenn das jeweilige Schema den zulässigen Wertebereich zusätzlich auf -9007199254740991 bis 9007199254740991 begrenzt.	`772107100456824`
string	`decimal-String`	Vorzeichenbehaftete Dezimalnummer	„`3.141592653589793238462643383279`“
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	“P1DT6H4S”
string	period		“`2019-07-30T06:43:40.252Z/PT3H`”
string	password		“`secret`”
string	email	RFC 5322	“`example@example.de`”
string	idn-email	RFC 6531	“`hello@buecher.example`”
string	hostname	RFC 1034	“`www.test.de`”
string	idn-hostname	RFC 5890	“`buecher.example`”
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-reference`	RFC 3986	“`/clothing/`”

JSON Data Type	Value	Spezifikation	Beispiel
string	uri-template	RFC 6570	"/users/{id}"
string	iri	RFC 3987	"https://buecher.example/"
string	iri-reference	RFC 3987	"/buecher-sport/"
string	uuid	RFC 9562	"018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21"
string	json-pointer	RFC 6901	"/items/0/id"
string	relative-json-pointer	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 NOT 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 NOT Fehlt die ausdrückliche Zulassung des Typs `null`, DARF die Eigenschaft NICHT mit dem Wert `null` übertragen werden.

SHOULD Eine optionale Eigenschaft, die nicht befüllt wird, 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` MÜSSEN fachlich und technisch unterschiedlich behandelt werden.

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.

MAY 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 NOT 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 NOT 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 NOT Das interne Signaturobjekt DARF NICHT als fachliches Nachrichtenformat verwendet werden und DARF NICHT als HTTP-Body übertragen werden.

Die Einzelheiten der signaturrelevanten JSON-Repräsentation werden in Teil 2, Kapitel 2.3 (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 MÜSSEN zurückgewiesen werden.

SHOULD 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.

MUST Das `ErrorResponse`-Schema MUSS mindestens die folgenden Eigenschaften enthalten:

- `code` (`string`): maschinenlesbarer, in der API-Spezifikation definierter Fehlercode,
- `title` (`string`): kurze, sprachunabhängige Bezeichnung des Fehlers,
- `detail` (`string`, optional): menschenlesbare Beschreibung des konkreten Fehlerfalls,
- `transactionId` (`string`): Effektive Transaktions-ID des abgelehnten Requests (bei einem Retry die `H2-Initial-Transaction-Id`, andernfalls die `H2-Transaction-Id` des Requests). Der Wert entspricht der `H2-Reference-Id` einer signaturpflichtigen synchronen Response. Kann die effektive Transaktions-ID wegen einer fehlenden oder fehlgeschlagenen Signaturprüfung nicht vertrauenswürdig bestimmt werden, MUSS `transactionId` abweichend hiervon die aktuelle `H2-Transaction-Id` des abgelehnten Übertragungsversuchs enthalten.

MAY API-Spezifikationen KÖNNEN das `ErrorResponse`-Schema um weitere Eigenschaften ergänzen.

MUST Kann wegen einer fehlenden oder syntaktisch ungültigen `H2-Transaction-Id` keine gültige `transactionId` bestimmt werden, MUSS die negative HTTP-Response ohne JSON-Body erfolgen.

MUST NOT Eine solche Response DARF weder eine `ErrorResponse` noch eine `H2-Reference-Id` enthalten. Sie ist ausschließlich als technischer Fehler zu behandeln und wird nicht auf Inhaltsdatenebene signiert.

4.5. Bestandteile eines API-Webdienstes

4.5.1. Namenskonvention für HTTP-Header

MUST Für neu definierte, proprietäre HTTP-Header MUSS ein einheitlicher, fachlich sprechender Namespace verwendet werden.

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

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

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

SHOULD Die Header SOLLTEN in der Spezifikation in einer einheitlichen kanonischen Schreibweise angegeben werden. 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 MÜSSEN die folgenden verbindlichen Header verwendet werden:

Kontext	Verbindliche Header
Jeder Request	`H2-Transaction-Id`, `H2-Message-Sender`, `H2-Message-Sender-Role`, `H2-Message-Receiver`, `H2-Message-Receiver-Role`, `H2-Business-Process`
Retry-Request	zusätzlich `H2-Initial-Transaction-Id`
Synchrone Response ohne Body	`H2-API-Version`; `H2-Reference-Id`, sofern eine eindeutige Bezugnahme auf eine Anfrage erforderlich ist
Fachlich relevante (signaturpflichtige) synchrone Response mit JSON-Body	`H2-Transaction-Id`, `H2-Reference-Id`, `H2-API-Version`, `H2-Signature`, `H2-Digest`, `H2-Cert` und `Content-Type`
Nicht signaturpflichtige synchrone Response mit JSON-Body	`H2-Transaction-Id`, `H2-Reference-Id`, `H2-API-Version` und `Content-Type`. Die Header `H2-Signature`, `H2-Digest` und `H2-Cert` DÜRFEN NICHT enthalten sein.
Asynchrone fachliche Response	`H2-Transaction-Id`, `H2-Reference-Id`, `H2-Message-Sender`, `H2-Message-Sender-Role`, `H2-Message-Receiver`, `H2-Message-Receiver-Role`, `H2-Business-Process`, `H2-API-Version`

Kontext	Verbindliche Header
Durch den Data Hub weitergeleiteter Request	zusätzlich `H2-Forwarded-By`

MUST Signaturpflichtige asynchrone fachliche Responses tragen zusätzlich `H2-Signature`, `H2-Digest` und `H2-Cert` gemäß Teil 2, Kapitel 2.3. `H2-API-Version` dokumentiert in der asynchronen Response die Version des antwortenden Dienstes und bleibt von der Ignorier-Regel für Requests (Kapitel 4.2) unberührt.

MUST Jede synchrone HTTP-Response mit JSON-Body MUSS eine neu erzeugte `H2-Transaction-Id` und eine `H2-Reference-Id` enthalten.

MUST Die `H2-Transaction-Id` einer signaturpflichtigen synchronen Response MUSS unmittelbar bei der Erzeugung der Response als UUID Version 7 neu erzeugt werden und identifiziert ausschließlich diese Response auf der aktuellen Kommunikationsstrecke.

MUST Die `H2-Reference-Id` einer synchronen Response MUSS auf die effektive Transaktions-ID des konkret beantworteten Requests verweisen: bei einem erstmaligen Request auf dessen `H2-Transaction-Id`, bei einem Retry auf dessen `H2-Initial-Transaction-Id`. Kann die effektive Transaktions-ID wegen einer fehlenden oder fehlgeschlagenen Signaturprüfung nicht vertrauenswürdig bestimmt werden, MUSS `H2-Reference-Id` abweichend hiervon die syntaktisch gültige aktuelle `H2-Transaction-Id` des abgelehnten Übertragungsversuchs enthalten. Liegt keine syntaktisch gültige `H2-Transaction-Id` vor, DARF `H2-Reference-Id` nicht enthalten sein; die Response MUSS ohne JSON-Body erfolgen.

MUST Synchrone negative Responses mit einem `ErrorResponse`-Body MÜSSEN signiert werden, sofern sie fachlich relevant sind, d. h. eine inhaltliche fachliche Ablehnung tragen (typischerweise `422 Unprocessable Content` oder ein `409 Conflict` aus fachlicher Prüfung). Rein technische Fehler-Responses (insbesondere 400, 401, 403, 404, 412, 415, 428, 429 sowie ein 409 aus Idempotenz- oder Vorbedingungsprüfung) sind nicht signaturpflichtig; die Abgrenzung richtet sich nach Teil 2, Kapitel 2.3.

Für negative Responses vorgelagerter Infrastrukturkomponenten ohne Zugriff auf den Signaturschlüssel gilt die Ausnahme gemäß Teil 2, Kapitel 2.3.

MUST NOT Synchrone Responses ohne HTTP-Body DÜRFEN die Header `H2-Signature`, `H2-Digest` und `H2-Cert` NICHT enthalten. Dies gilt insbesondere für `202 Accepted` und `204 No Content`, soweit kein Response-Body übertragen wird.

MUST NOT Die HTTP-Response `101 Switching Protocols` sowie weitere Responses, die ausschließlich zum Aufbau einer Kommunikationsverbindung gehören und in der jeweiligen technischen Schnittstellenbeschreibung ausdrücklich ausgenommen sind, DÜRFEN NICHT auf Inhaltsdatenebene signiert werden.

MUST Die Headerwerte MÜSSEN die folgenden Formate einhalten:

Zusätzliche Anforderungen

MUST Die `H2-Transaction-Id` MUSS unmittelbar bei der Erzeugung jedes Requests, jeder asynchronen fachlichen Response und jeder signaturpflichtigen synchronen Response neu erzeugt werden.

Der in der UUID Version 7 enthaltene Unix-Zeitwert beschreibt den technischen Erzeugungszeitpunkt der `H2-Transaction-Id`.

MUST NOT Der enthaltene Zeitwert DARF NICHT als fachlicher Zeitstempel verwendet werden.

MUST Fachlich relevante Zeitpunkte MÜSSEN weiterhin in den dafür vorgesehenen fachlichen Eigenschaften übertragen werden.

MUST Requests mit einem für den adressierten API-Webdienst nicht zulässigen Wert in `H2-Business-Process` MÜSSEN mit HTTP `400 Bad Request` und einer `ErrorResponse` abgelehnt werden.

`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-Sender-Role`

- OpenAPI Typ: `string`
- Zweck: Angabe der Marktpartner-Rolle
- Zulässige Werte:
- `GridOperator`
- `MarketAreaManager`
- `BalanceGroupResponsibleParty`
- `TransportCustomer`
- `DataHub`
- `StorageOperator`
- Beispiel: `GridOperator`

`H2-Message-Receiver`

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

- Beispiel: `9871000123456`

`H2-Message-Receiver-Role`

- OpenAPI Typ: `string`
- Zweck: Angabe der Marktpartner-Rolle
- Zulässige Werte:
- `GridOperator`
- `MarketAreaManager`
- `BalanceGroupResponsibleParty`
- `TransportCustomer`
- `DataHub`
- `StorageOperator`
- Beispiel: `MarketAreaManager`

`H2-Transaction-Id`

- OpenAPI Typ: `string`
- OpenAPI Format: uuid
- Zweck: Eindeutige Identifikation einer einzelnen HTTP-Nachricht auf der aktuellen Kommunikationsstrecke; bei signaturpflichtigen Nachrichten zugleich Grundlage von Zeitfenster- und Replay-Prüfung, beim WebSocket-Upgrade zugleich connectionId.
- 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`

`H2-Initial-Transaction-Id`

- OpenAPI Typ: `string`
- OpenAPI Format: uuid
- Zweck: Referenzierung des erstmaligen Aufrufs und Unterstützung der idempotenten Verarbeitung von Retries.
- 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`

Der in der `H2-Initial-Transaction-Id` enthaltene Zeitwert beschreibt den Erzeugungszeitpunkt des initialen Aufrufs und NICHT den Zeitpunkt des aktuellen Retry-Requests.

`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.
- Zusätzliche Anforderung: Die zulässigen Werte werden je fachlichem Prozess durch die API-Governance in der Anlage zu dieser Guideline festgelegt und verwaltet. Werte werden in `lowerCamelCase` gebildet.
- Pattern: ^[a-z][a-zA-Z0-9]{2,63}$
- 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`

`H2-Forwarded-By`

- OpenAPI Typ: `string`
- Zweck: Kennzeichnung des Data Hub als technische weiterleitende Stelle
- Zusätzliche Anforderung: Der Wert MUSS die 13-stellige Marktpartner-ID des Data Hub enthalten.
- Pattern: ^[0-9]{13}$
- Beispiel: `9871000123456`

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-Sender-Role: GridOperator`
- `H2-Message-Receiver: 9871000654321`
- `H2-Message-Receiver-Role: MarketAreaManager`
- `H2-Business-Process: nominationSubmission`
- `H2-Forwarded-By: 9871000000000`

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-Sender-Role: GridOperator
- H2-Message-Receiver: 9871000654321
- H2-Message-Receiver-Role: MarketAreaManager
- H2-Business-Process: nominationSubmission
- H2-Forwarded-By: 9871000000000

MUST Wenn sich eine Antwort auf eine Anfrage beziehen MUSS, gilt für H2-Reference-Id:

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

Der in der H2-Reference-Id enthaltene Zeitwert beschreibt den Erzeugungszeitpunkt der referenzierten Nachricht und NICHT den Erzeugungszeitpunkt der aktuellen Nachricht.

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 Bei einem Retry MUSS der Client eine neue H2-Transaction-Id vergeben.

MUST Zusätzlich MUSS der Client in H2-Initial-Transaction-Id die H2-Transaction-Id des erstmaligen Aufrufs übertragen.

Die H2-Initial-Transaction-Id ermöglicht die Identifikation zusammengehöriger Aufrufe. Die tatsächliche idempotente Verarbeitung wird durch die in diesem Kapitel festgelegten serverseitigen Prüf- und Speichermechanismen sichergestellt.

MUST NOT Ist eine Anfrage kein Retry, 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. sofern

vorhanden die „H2-Initial-Transaction-Id“ zurückgegeben werden, um eine eindeutige Zuordnung zur ursprünglichen Anfrage zu ermöglichen.

MUST Eine signaturpflichtige synchrone Response MUSS eine eigene H2-Transaction-Id führen. Ihre H2-Reference-Id MUSS die effektive Transaktions-ID des konkret beantworteten Requests enthalten.

MUST NOT Die eigene H2-Transaction-Id der synchronen Response DARF NICHT mit der H2-Transaction-Id oder H2-Initial-Transaction-Id des Requests gleichgesetzt werden.

Für die Idempotenzprüfung gilt als effektive Transaktions-ID:

- bei einem erstmaligen Request die H2-Transaction-Id,
- bei einem Retry die H2-Initial-Transaction-Id.

Die effektive Transaktions-ID dient der fachlichen Idempotenz. Davon zu unterscheiden ist die tatsächliche H2-Transaction-Id des einzelnen Übertragungsversuchs, die für den technischen Replay-Schutz verwendet wird.

MUST Der API-Anbieter MUSS jede tatsächliche H2-Transaction-Id innerhalb des in der JWS-Spezifikation festgelegten Signaturzeitfensters je Umgebung und technischem Signierer gemäß Kapitel 2.3.4 auf Wiederverwendung prüfen.

MUST Die Prüfung und Reservierung des Replay-Schlüssels MUSS atomar und bei horizontal skalierter Verarbeitung clusterweit wirksam sein. Der Replay-Schlüssel MUSS mindestens aus Umgebung, MP-ID des technischen Signierers und H2-Transaction-Id bestehen.

MUST Eine wiederverwendete H2-Transaction-Id DARF keine erneute fachliche Ausführung auslösen. Bei identischen Vergleichsmerkmalen gelten die nachfolgenden Idempotenzregeln; bei abweichenden Vergleichsmerkmalen MUSS die Anfrage mit 409 Conflict zurückgewiesen werden.

MUST Kann die atomare Replay-Prüfung nicht verlässlich durchgeführt werden, MUSS der API-Anbieter die Anfrage ohne fachliche Verarbeitung mit 503 Service Unavailable ablehnen.

MUST Der API-Anbieter MUSS einen Retry anhand der Kombination aus H2-Message-Sender und effektiver Transaktions-ID identifizieren.

MUST Der API-Anbieter MUSS zu dieser Kombination mindestens die folgenden Vergleichsmerkmale speichern:

- H2-Message-Receiver,
- H2-Business-Process,
- HTTP-Methode,
- normalisierte URI einschließlich Query-Parametern und
- Hashwert des Request-Bodys.

- `If-Match` oder `If-None-Match` einschließlich des Falls, dass der jeweilige Header nicht vorhanden ist.

MUST Der erneute Aufbau einer beendeten oder abgebrochenen zustandsbehafteten Verbindung (insbesondere ein WebSocket-Upgrade nach einem Verbindungsabbruch) MUSS als neuer Aufruf mit neu erzeugter `H2-Transaction-Id` behandelt werden. Er ist kein Retry-Request im Sinne dieses Kapitels.

MUST NOT `H2-Initial-Transaction-Id` DARF bei einem solchen erneuten Verbindungsaufbau NICHT übertragen werden.

MUST Der Hashwert eines JSON-Request-Bodys MUSS über die gemäß RFC 8785 kanonisierte JSON-Repräsentation gebildet werden.

MUST Bei einem anhand der effektiven Transaktions-ID erkannten Retry MUSS die Idempotenzprüfung vor einer erneuten Prüfung von `If-Match` oder `If-None-Match` und vor der fachlichen Verarbeitung erfolgen.

MUST NOT Wird dieselbe effektive Transaktions-ID mit identischer HTTP-Methode, identischer URI und identischem Request-Body erneut verwendet, DARF der fachliche Vorgang NICHT erneut ausgeführt werden.

MUST In diesem Fall MUSS der API-Anbieter entweder das bereits ermittelte Ergebnis oder den Verarbeitungsstatus der ursprünglichen Anfrage zurückgeben.

MUST Die Vorbedingung `If-Match` oder `If-None-Match` DARF beim Zurückgeben dieses Ergebnisses NICHT erneut gegen den aktuellen Ressourcenstand geprüft werden. Wurde die ursprüngliche Anfrage mit HTTP 412 Precondition Failed beantwortet, MUSS auch dieses Ergebnis beim Retry unverändert erneut ausgeliefert werden.

MUST Wird dieselbe effektive Transaktions-ID mit einer abweichenden HTTP-Methode, einer abweichenden URI, einem abweichenden Request-Body oder einem abweichenden Wert beziehungsweise abweichenden Vorhandensein von `If-Match` oder `If-None-Match` verwendet, MUSS die Anfrage mit HTTP 409 Conflict zurückgewiesen werden.

MUST Zeitgleich eintreffende Requests mit derselben effektiven Transaktions-ID MÜSSEN so verarbeitet werden, dass der fachliche Vorgang höchstens einmal ausgeführt wird.

MUST Die Aufbewahrungsdauer der für die Idempotenzprüfung erforderlichen Informationen MUSS mindestens den maximal zulässigen Retry-Zeitraum zuzüglich der maximalen Verarbeitungsdauer des fachlichen Vorgangs umfassen.

MUST Die konkrete Aufbewahrungsdauer MUSS in der jeweiligen API-Spezifikation oder Prozessbeschreibung festgelegt werden.

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

Hinweis: Die Abbildung veranschaulicht ausschließlich die Vergabe und Referenzierung der Transaktionskennungen. Der in Kapitel 2.3 festgelegte zentrale Kommunikationsweg über den Data Hub bleibt hiervon unberührt.

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 NOT GET-Requests DÜRFEN KEINEN Request-Body enthalten.

MUST GET DARF 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 wird.

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.

Für ausdrücklich spezifizierte technische Protokoll-Upgrades sind zusätzlich die Statuscodes `101` Switching Protocols und `426` Upgrade Required gemäß der jeweiligen technischen Schnittstellenbeschreibung zulässig.

4.7.1. Positive Response Codes

Code	Name	Erläuterung
101	Switching Protocols	Der technische Wechsel auf das in der Schnittstellenbeschreibung vereinbarte Protokoll wurde erfolgreich durchgeführt. Die Response enthält keinen JSON-Body und wird nicht auf Inhaltsdatenebene signiert.
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
304	Not Modified	Bedingte `GET`-Anfrage mit `If-None-Match`: Der Verzeichniseintrag ist gegenüber dem angegebenen `ETag` unverändert; es wird kein Response-Body übertragen.

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.

Code	Name	Erläuterung
401	Unauthorized	Die HTTP-Anfrage konnte auf Anwendungsebene nicht erfolgreich authentisiert werden. Scheitert bereits die mTLS-Authentisierung, kommt keine HTTP-Anfrage zustande und es wird keine HTTP-Response zurückgegeben.
403	Forbidden	Die Authentisierung war erfolgreich, der authentisierte Marktpartner 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	Methode nicht zulässig; `Allow`-Header nennt zulässige Methoden.
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.
410	Gone	Ressource/API-Version dauerhaft entfernt (z. B. `retired`).
412	Precondition Failed	Eine mit `If-Match` oder `If-None-Match` übertragene Vorbedingung ist nicht erfüllt. Technische Response, nicht signaturpflichtig (Teil 2, Kapitel 2.3).
415	Unsupported Media Type	Medientyp in der Anfrage ist nicht unterstützt
422	Unprocessable Content	JSON Schema wurde verletzt.
428	Precondition Required	Eine für die Operation verbindlich vorgeschriebene Vorbedingung `If-Match` oder `If-None-Match` fehlt.
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

Code	Name	Erläuterung
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 NOT 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.

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 Leach, P. Universally *Unique IDentifiers (UUIDs).* IETF RFC 9562. May 2024. https://www.rfc-editor.org/rfc/rfc9562

[OpenAPI 3.1] OpenAPI Initiative. OpenAPI Specification Version 3.1.2. https://spec.openapis.org/oas/v3.1.2.html

[JSON Schema 2020-12] JSON Schema Organization. JSON Schema Draft 2020-12, Core and Validation Specifications. Published 16 June 2022. https://json-schema.org/draft/2020-12

[SemVer 2.0.0] Preston-Werner, T. Semantic Versioning 2.0.0. https://semver.org/spec/v2.0.0.html

[BNetzA BK7-24-01-014] Bundesnetzagentur. Beschluss im Festlegungsverfahren WasABi in Sachen Wasserstoff-Ausgleichs- und Bilanzierungsgrundmodell vom 27.10.2025. https://www.bundesnetzagentur.de/DE/Beschlusskammern/1_GZ/BK7-GZ/2024/BK7-24-0014/BK7-24-01-0014_Beschluss_Internet.html

[BNetzA BK7-24-01-015] Bundesnetzagentur. Beschluss im Festlegungsverfahren WaKandA in Sachen Wasserstoff-Kapazitäten-Grundmodell und Abwicklung des Netzzugangs vom 27.10.2025. https://www.bundesnetzagentur.de/DE/Beschlusskammern/1_GZ/BK7-GZ/2024/BK7-24-0015/BK7-24-01-0015_Beschluss_Internet.html

[BDEW API 1.0a] BDEW. API-Guideline. Version 1.0a, 01.10.2024. https://bdew-mako.de/pdf/API_Guideline_1_0a_20241001.pdf

Teil 2
–
Regelungen zur Absicherung der Kommunikation für API-Webdienste im deutschen Wasserstoff-markt

1. Einleitung

Dieser Teil der Guideline beschreibt die Anforderungen und Rahmenbedingungen für die Absicherung und Durchführung von API-basierter Kommunikation im Wasserstoffmarkt.

Er ergänzt die in Teil 1 dieser Guideline beschriebenen fachlichen und technischen Vorgaben für API-Webdienste um die Anforderungen an die sichere Übertragung, Authentisierung, Integritätssicherung sowie die Auffindbarkeit von Kommunikationsendpunkten. Gemeinsam bilden beide Teile den verbindlichen Rahmen für die standardisierte API-Kommunikation im deutschen Wasserstoffmarkt.

Grundlage bilden die regulatorischen Vorgaben der Bundesnetzagentur (BNetzA)[1] sowie die technischen Richtlinien des Bundesamts für Sicherheit in der Informationstechnik (BSI), insbesondere die Technische Richtlinie TR-03116-3[2]. Demnach sind die dort definierten kryptographischen Verfahren verbindlich anzuwenden. Darüber hinaus ist gemäß § 52 Abs. 4 des Messstellenbetriebsgesetzes (MsbG)[3] die Nutzung der Smart-Metering-Public-Key-Infrastruktur (SM-PKI) des BSI vorzusehen.

Die in diesem Teil der Guideline beschriebenen Parameter sowie etwaige Abweichungen konkretisieren diese Vorgaben für den Anwendungsfall des Wasserstoffmarktes.

Inhaltlich basiert dieser Teil der Guideline auf den von der edi@energy-Arbeitsgruppe veröffentlichten Dokumenten „Regelungen zum Übertragungsweg für API-Webdienste“, Version 1.2[4], sowie „Regelungen zum Verzeichnisdienst“, Version 1.1[5]. Diese wurden für die spezifischen Anforderungen des Wasserstoffmarktes angepasst und weiterentwickelt.

Die in diesem Teil der Guideline beschriebenen Anforderungen sind von allen Marktrollen einzuhalten, die im Rahmen der Festlegungen WasABi und WaKandA API-Webdienste bereitstellen oder nutzen.

Die Kapitel 2 und 3 beschreiben die Anforderungen an die Absicherung der Kommunikationsverbindungen und orientieren sich dabei an den Regelungen für API-Webdienste [4]. Kapitel 4 behandelt die Nutzung und Einbindung des Verzeichnisdienstes im Wasserstoffmarkt.

---

[1] Vgl. Bundesnetzagentur, Beschluss BK7-24-01-014 (WasABi) vom 27.10.2025, Tenorziffer 7 lit. c) ii.
[2] Bundesamt für Sicherheit in der Informationstechnik (13.12.2024): BSI TR-03116-3 – Kryptographische Vorgaben für Projekte der Bundesregierung, Teil 3: Intelligente Messsysteme, Stand 2025.
[3] Messstellenbetriebsgesetz (MsbG), § 52 Abs. 4, in der jeweils geltenden Fassung.
[4] BDEW, Regelungen zum Übertragungsweg für API-Webdienste, Version 1.2, Publikationsdatum 01.04.2025, anzuwenden ab 01.10.2025.
[5] BDEW, Regelungen zum Verzeichnisdienst, Version 1.1, Publikationsdatum 01.04.2025, anzuwenden ab 01.10.2025.

2. Technische Vorgaben der API-Webdienste

2.1. Netzwerkebene

MUST Der Zugriff auf API-Webdienste MUSS über DNS-Namen erfolgen.

MUST Die API-Webdienste MÜSSEN über IPv4 erreichbar sein.

MUST Der API-Nutzer MUSS eine gesicherte TLS-Verbindung gemäß TR-03116-3 über IPv4 zum API-Webdienst herstellen.

SHOULD Die beschriebenen API-Webdienste SOLLTEN IPv6 unterstützen.

MUST NOT Der Zugriff auf die API-Webdienste DARF NICHT über IP-Adressen erfolgen.

Beispiele:

- zulässiger Aufruf am Data Hub:
  `https://h2-datahub.example/9850123400004/H2API00001/1`
- zulässiger Aufruf eines API-Endpunkts vom Data Hub aus:
  `https://9850123400004.h2-cloud.example/v1/allocations`
- unzulässig: http://127.0.0.1/v1/allocations

2.2. Transportebene

MUST Für den Aufbau der HTTPS-Verbindung MUSS das TLS-Protokoll mindestens in der Version 1.2 gemäß den Vorgaben der Technischen Richtlinie TR-03116-3 eingesetzt werden, insbesondere gemäß Kapitel „TLS-Kommunikation im WAN und Inhaltsdatensicherung in der Marktkommunikation“.

MUST Für den Aufbau der TLS-Verbindung MUSS die Erweiterung Server Name Indication (SNI) gemäß RFC 6066 unterstützt und verwendet werden.

MUST Soweit die TLS-Erweiterung `record_size_limit` verwendet wird, MUSS deren Verarbeitung RFC 8449 entsprechen.

MUST Die Kommunikationspartner MÜSSEN sich gegenseitig mittels X.509-Zertifikaten der Smart-Metering-PKI authentifizieren.

MUST Die TLS-Verbindung MUSS als Mutual TLS (mTLS) gemäß den Vorgaben der Smart-Metering-PKI aufgebaut werden.

MUST Für die mTLS-Authentisierung MUSS jeweils das TLS-Zertifikat des gültigen `EMT.API`-Zertifikatstripels verwendet werden.

2.3. Inhaltsdatensicherungsebene

MUST Alle fachlichen HTTP-Requests im Sinne dieser Guideline MÜSSEN signiert werden, unabhängig davon, ob sie einen Payload enthalten. HTTP-Requests, die ausschließlich dem Aufbau einer Kommunikationsverbindung dienen (insbesondere der HTTP-Upgrade-Request zum Aufbau einer WebSocket-Verbindung gemäß Kapitel 5.2), sind hiervon ausgenommen; ihre Absicherung erfolgt durch die mTLS-Authentisierung gemäß Kapitel 2.2.

MUST Synchrone HTTP-Responses MÜSSEN signiert werden, wenn sie fachlich relevant sind. Fachlich relevant ist eine Response, die einen Verzeichniseintrag oder einen anderen signierten Datenbestand zurückgibt oder eine fachliche Zustandsänderung verbindlich bestätigt oder ablehnt. Eine fachlich relevante Response wird unabhängig davon signiert, ob sie einen positiven oder einen `ErrorResponse`-Body trägt.

MUST NOT Rein technische oder transportbezogene Responses DÜRFEN NICHT auf Inhaltsdatenebene signiert werden. Hierzu gehören insbesondere Responses mit den Statuscodes 400, 401, 403, 404, 412, 415, 428 und 429 sowie bodylose Responses wie `204 No Content` und `304 Not Modified`. Maßgeblich ist nicht der Statuscode allein, sondern ob die Response eine fachliche Aussage trifft: Trägt sie einen fachlichen `ErrorResponse`-Body mit einer inhaltlichen Ablehnung (typischerweise `409 Conflict` aus fachlicher Prüfung oder `422 Unprocessable Content`), ist sie fachlich relevant und MUSS signiert werden; ein 409 aus reiner Idempotenz- oder Vorbedingungsprüfung sowie ein 412 aus fehlgeschlagenem optimistischem Sperren sind technisch und werden NICHT signiert.

MUST Die jeweilige API-Spezifikation MUSS je Operation ausweisen, welche Responses fachlich relevant und damit signaturpflichtig sind. Ist eine Response nicht ausdrücklich als fachlich relevant gekennzeichnet, gilt sie als technisch und wird NICHT signiert.

MUST NOT Aus einer nicht signierten technischen Response DARF ein Client keine fachliche Endgültigkeit ableiten; der zugrunde liegende Vorgang gilt als technisch fehlgeschlagen und offen und ist gemäß Teil 1, Kapitel 3.4 zu behandeln.

MUST Die Signaturpflicht gilt für applikativ durch den API-Webdienst erzeugte Responses. Vorgelagerte Infrastrukturkomponenten ohne Zugriff auf den Signaturschlüssel des API-Anbieters DÜRFEN keine fachlich relevanten Responses und insbesondere keine fachlichen Ablehnungen mit `ErrorResponse`-JSON-Body erzeugen; von ihnen erzeugte technische Fehler-Responses MÜSSEN ohne Body oder mit einem `Content-Type` ungleich `application/json` erfolgen. Einzelheiten regelt die JWS-Spezifikation.

Solche Infrastruktur-Responses enthalten kein Feld `transactionId`; ihre Korrelation zu einem Request erfolgt ausschließlich über die mitgeführten `H2-Header`. Ein Client DARF das Vorhandensein eines `transactionId`-Feldes bei solchen Responses NICHT voraussetzen.

MUST NOT Synchrone HTTP-Responses ohne Body DÜRFEN NICHT auf Inhaltsdatenebene signiert werden.

Aus einer nicht signierten Response DARF ein Client keine fachliche Endgültigkeit ableiten (Teil 2, Kapitel 2.3).

MUST NOT Der HTTP-Upgrade-Request zum Aufbau einer WebSocket-Verbindung und sämtliche zu diesem Upgrade gehörenden HTTP-Responses, einschließlich `101 Switching Protocols`, DÜRFEN NICHT auf Inhaltsdatenebene signiert werden. Ihre Absicherung erfolgt ausschließlich durch die mTLS-Authentisierung und die Prüfungen der WebSocket-Spezifikation.

Bei Nachrichten ohne Payload umfasst die Signatur die übrigen Signaturbestandteile gemäß diesem Kapitel.

MUST NOT Das Fehlen eines Payloads DARF NICHT mit einem übertragenen leeren JSON-Objekt gleichgesetzt werden.

MUST Die in TR-03116-3 beschriebenen kryptografischen Verfahren (Hash- und Signatur-Algorithmus) sowie die Hinweise zur Umsetzung MÜSSEN angewendet werden.

MUST Für die Inhaltsdatensignatur MUSS das `SIG`-Zertifikat desselben gültigen `EMT.API`-Zertifikatstripels verwendet werden. Der Header `H2-Cert` MUSS dieses Signaturzertifikat enthalten.

MUST NOT Die fachlichen Nutzdaten DÜRFEN auf Anwendungsebene NICHT zusätzlich verschlüsselt werden, sofern die jeweilige API-Spezifikation dies nicht ausdrücklich vorsieht.

MUST Folgende Bestandteile MÜSSEN signiert werden:

- Die adressierte Ressource (URI) inklusive aller Query-Parameter
- Folgende Header sind Bestandteil der Signaturbildung, sofern sie im jeweiligen Nachrichtenkontext vorhanden sind:
  - `H2-Transaction-Id`
  - `H2-Initial-Transaction-Id`
  - `H2-Reference-Id`
  - `H2-Message-Sender`
  - `H2-Message-Sender-Role`
  - `H2-Message-Receiver`
  - `H2-Message-Receiver-Role`
  - `H2-Business-Process`
  - `H2-Forwarded-By`
  - `H2-API-Version`
  - `Content-Type`
  - `If-Match`
  - `If-None-Match`
  - `ETag`
- Payload (JSON-Objekt)

MUST Zusätzlich MÜSSEN die HTTP-Methode und bei Responses der HTTP-Statuscode Bestandteil der Signaturbildung sein.

MUST Bei einer signaturpflichtigen synchronen Response MÜSSEN zusätzlich die normalisierte URI und die HTTP-Methode des konkret beantworteten Requests, der HTTP-Statuscode der Response, die eigene `H2-Transaction-Id` der Response, die `H2-Reference-Id`, `H2-API-Version`, `Content-Type`, ein übertragener `ETag` und der tatsächlich übertragene Response-Body Bestandteil der Signaturrepräsentation sein.

MUST Die Werte für URI, HTTP-Methode und Referenz auf den beantworteten Request MÜSSEN aus dem lokal vorliegenden Request-Kontext gebildet werden. Nicht vertrauenswürdige oder nur aus der Response abgeleitete Ersatzwerte DÜRFEN hierfür NICHT verwendet werden.

MUST Der signierende API-Anbieter MUSS bei der Response-Erzeugung über den vollständigen Request-Kontext verfügen. Bei asynchroner oder prozessübergreifender Verarbeitung MUSS dieser Kontext vertrauenswürdig bis zur Signaturerstellung durchgereicht werden; andernfalls MUSS die Verarbeitung fehlschlagen.

MUST Der technische Erstellungszeitpunkt einer signaturpflichtigen HTTP-Nachricht MUSS aus dem 48-Bit-Unix-Zeitwert der aktuellen `H2-Transaction-Id` gemäß UUID Version 7 bestimmt werden. Ein zusätzliches Feld `created` oder ein zusätzlicher Signaturzeit-Header DARF für HTTP-Nachrichten NICHT verwendet werden.

MUST Bei einem Retry MUSS der Zeitwert der aktuellen `H2-Transaction-Id` ausgewertet werden; der Zeitwert der `H2-Initial-Transaction-Id` ist hierfür nicht maßgeblich. Bei einer synchronen oder asynchronen Response ist der Zeitwert ihrer eigenen `H2-Transaction-Id` maßgeblich; `H2-Reference-Id` dient ausschließlich der Korrelation.

MUST Die JWS-Spezifikation MUSS das zulässige Alter und den maximalen Zukunftsversatz des aus der UUID Version 7 abgeleiteten Zeitwerts sowie die daraus folgende Mindestaufbewahrung des Replay-Schlüssels verbindlich festlegen.

MUST Alle Systeme, die signaturpflichtige Nachrichten erzeugen oder prüfen, MÜSSEN ihre Systemzeit über eine zuverlässige Zeitquelle (z. B. NTP) synchronisieren; die Uhrenabweichung MUSS deutlich unterhalb des maximalen Zukunftsversatzes liegen.

MUST Die URI MUSS nach RFC 3986 normalisiert werden; vor der Berechnung des JSON-Objekt-Hashwerts MUSS das JSON-Objekt gemäß RFC 8785 kanonisiert werden.

MUST Die Normalisierung und Signaturprüfung MÜSSEN auf Basis des unveränderten HTTP-Request-Targets erfolgen, bevor ein Proxy, Gateway, Router oder Webframework Pfad oder Query-String dekodiert, umschreibt, sortiert oder anderweitig normalisiert.

MUST NOT JSON-Objekte mit mehrfach vorkommenden Eigenschaftsnamen sowie HTTP-Nachrichten mit mehrfach übertragenen signaturrelevanten Headerfeldern DÜRFEN NICHT signiert oder fachlich verarbeitet werden. Die Einzelheiten legt die JWS-Spezifikation fest.

MUST Zur Berechnung der Signatur MUSS JSON Web Signature (JWS) verwendet werden.

MUST Die konkrete JWS-Serialisierung, die Übertragung der Signatur, die Referenzierung des Signaturzertifikats sowie die eindeutige Bildung der zu signierenden Repräsentation MÜSSEN in der technischen Schnittstellenspezifikation festgelegt werden.

MUST Die technische Schnittstellenspezifikation MUSS insbesondere die Behandlung der URI, der Query-Parameter, der signaturrelevanten HTTP-Header, von Nachrichten ohne Payload sowie optionaler Signaturbestandteile eindeutig festlegen.

MUST Payload und Steuerinformationen MÜSSEN zur Signaturbildung in einem JSON-Objekt zusammengeführt werden. Dieses Objekt dient ausschließlich der Signaturerzeugung und wird nicht übertragen.

Die Vertraulichkeit der Kommunikation wird durch die TLS-gesicherte Transportverbindung gewährleistet.

Beispiel der internen, ausschließlich zur Signaturbildung erzeugten JSON-Repräsentation eines Requests, dessen Payload ein leeres JSON-Objekt ist:

```
{
   "meta":
     {
          "uri": "/9766587654321/H2API00001/1",
                 "method": "POST",
          "H2-Message-Sender": "9871000123456",
          "H2-Message-Sender-Role": "GridOperator",
          "H2-Message-Receiver": "9871000654321",
          "H2-Message-Receiver-Role": "DataHub",
          "H2-Transaction-Id": "018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21",
          "H2-Business-Process": "preliminaryMeasurementSubmission",
          "Content-Type": "application/json"
     },
    "data":
    { }
}
```

Hinweis: Die dargestellte Struktur wird nicht übertragen; über HTTP wird ausschließlich der Payload (hier: { }) im Request-Body übermittelt. Die Elemente unter `meta` entsprechen den signaturrelevanten Headern und Steuerinformationen gemäß diesem Kapitel.

Beispiel der internen JSON-Repräsentation eines Requests ohne Payload:

```
{
   "meta":
     {
```

```
        "uri": "/9766587654321/H2API00001/1",
               "method": "POST",
        "H2-Message-Sender": "9871000123456",
        "H2-Message-Sender-Role": "GridOperator",
        "H2-Message-Receiver": "9871000654321",
        "H2-Message-Receiver-Role": "DataHub",
        "H2-Transaction-Id": "018f0d4e-6b7a-7c31-b5c2-8d4d0d8a3f21",
        "H2-Business-Process": "preliminaryMeasurementSubmission"
    }
}
```

Hinweis: Bei Nachrichten ohne Payload entfällt das Element `data` ersatzlos.

2.4. Prod-/Test-System

Folgendes Szenario gilt ab dem 01.01.2028:

MUST Es MÜSSEN getrennte Systemumgebungen für den Produktivbetrieb (PROD) und für Test- und Abnahmezwecke (TEST) eingeplant werden. Das TEST-System MUSS dauerhaft erreichbar und funktionsfähig betrieben werden, sodass angebundene Parteien jederzeit die jeweils aktuellste sowie zukünftige API-Versionen integrieren, testen und validieren können.

MUST Neue oder geänderte API-Versionen MÜSSEN vor dem produktiven Einsatz ausschließlich im TEST-System bereitgestellt werden. Das TEST-System MUSS dabei in Bezug auf Schnittstellenstruktur, Kommunikationsprotokolle, Sicherheitsmechanismen, Authentisierung und Autorisierung sowie Datenformate dem PROD-System technisch entsprechen, soweit dies ohne Verwendung produktiver Daten möglich ist.

MUST Ein Zugriff auf das PROD-System DARF erst nach erfolgreicher technischer und fachlicher Validierung im TEST-System sowie nach expliziter Freigabe erfolgen.

MUST NOT Ein direkter Test neuer oder geänderter API-Versionen im PROD-System DARF NICHT erfolgen.

MUST Gemäß SM-PKI MÜSSEN für die Kommunikation in den Umgebungen PROD und TEST unterschiedliche SM-PKI-Zertifikate verwendet werden

MUST Das TEST-System MUSS grundsätzlich dauerhaft erreichbar und für die vorgesehenen Integrations-, Test- und Abnahmezwecke nutzbar sein. Geplante Wartungsarbeiten, Releases sowie zeitlich und sachlich begrenzte funktionale Einschränkungen sind zulässig, sofern sie angekündigt, auf das erforderliche Maß begrenzt und so durchgeführt werden, dass die Einführung und Validierung neuer API-Versionen weiterhin gewährleistet bleibt.

Für die Testphase vom 01.07.2027 bis 31.12.2027 gilt:

MUST Für die Testphase MUSS ein TEST System zur Verfügung gestellt werden.

MUST Die API-Kommunikation MUSS zunächst mit geeigneten TLS-Testzertifikaten außerhalb der SM-PKI getestet werden.

MUST Nach erfolgreichem Abschluss dieser Tests MUSS bis zum 31.12.2027 zusätzlich eine Testphase mit SM-PKI-Zertifikaten angeboten werden. Damit wird sichergestellt, dass die Beschaffung der SM-PKI-Zertifikate kein Hindernis für den Test der API-Funktionen darstellt.

3. Zertifikate und Smart Metering PKI

MUST Die Kommunikation MUSS durch Verwendung der Smart Metering PKI (SM-PKI) des BSI abgesichert werden.

MUST Die Vorgaben der Certificate Policy (CP) der SM-PKI[6] MÜSSEN eingehalten werden. Die Kommunikationspartner, API-Nutzer und Anbieter, sind nach dem Rollenkonzept der SM-PKI in der Rolle passiver EMT am Marktgeschehen beteiligt.

3.1. Vertrauensdienstanbieter

MUST Die Vertrauensdienstanbieter MÜSSEN eine Sub-CA-Instanz gemäß der CP der SM-PKI [10] sein.

3.2. Veröffentlichung von Kommunikationsmetadaten

MUST Jeder Marktpartner MUSS die Metadaten zu seinen verwendeten Zertifikaten, die API-Kennung und den zur API-Kennung zugehörigen Endpunkt (URL) der API am Data Hub veröffentlichen [11].

3.3. Zertifikate: Parameter und Anforderungen

MUST Die Zertifikatanforderungen MÜSSEN der CP der SM-PKI entsprechen. Jeder Kommunikationspartner MUSS über ein gültiges Zertifikatstripel der Klasse `EMT.API` verfügen.

Insbesondere gilt:

MUST Das `EMT.API`-Zertifikatstripel MUSS aus einem Signaturzertifikat `SIG`, einem Verschlüsselungszertifikat `ENC` und einem Zertifikat für den Aufbau des TLS-Kanals `TLS` bestehen.
MUST Für mTLS MUSS das `TLS`-Zertifikat verwendet werden. Für JWS-Inhaltsdatensignaturen und Signaturen von Verzeichniseinträgen MUSS das `SIG`-Zertifikat verwendet werden.
MUST NOT Das `ENC`-Zertifikat DARF im Rahmen dieser Guideline NICHT zur zusätzlichen Verschlüsselung fachlicher Nutzdaten auf Anwendungsebene verwendet werden; die Vertraulichkeit wird durch TLS gewährleistet.
MUST Die Zertifikate eines Tripels MÜSSEN derselben Marktpartner-ID zugeordnet sein. Der Common Name MUSS dem Schema `<Organisation>.EMT.API<Extension>` folgen; das Feld Organizational Unit des Subject MUSS die Marktpartner-ID enthalten.
MUST Der Subject Alternative Name MUSS eine URI des maßgeblichen Verzeichnisdienstes enthalten. Weitere Anforderungen ergeben sich aus der Certificate Policy der SM-PKI.

[6] Bundesamt für Sicherheit in der Informationstechnik, Certificate Policy der Smart Metering PKI, Version 1.1.2, 25.01.2023.

MUST Ein Zertifikatswechsel MUSS grundsätzlich für das vollständige Zertifikatstripel berücksichtigt werden, da die Zertifikate nach den Vorgaben der SM-PKI nicht einzeln beantragt oder ausgetauscht werden.
MAY Ein `EMT.API`-Zertifikatstripel KANN für einen oder mehrere API-Webdienste desselben Marktpartners genutzt werden.

3.4. Zertifikatswechsel

Dieses Kapitel beschreibt das Vorgehen, wenn ein Zertifikatswechsel nötig wird.

3.4.1. Zertifikatswechsel bei ablaufenden Zertifikaten

MUST Spätestens 5 Werktage, bevor Zertifikate ungültig werden, MUSS der Inhaber dieser Zertifikate die Nachfolgezertifikate zur Verfügung gestellt haben. Somit entsteht ein Überlappungszeitraum von mindestens 5 Werktagen, in dem die bisherigen und die neuen Zertifikate gleichzeitig gültig sind. Für diesen Zeitraum gilt: Innerhalb dieses Überlappungszeitraums kann bei allen Marktpartnern die Umstellung von den bisher genutzten auf die neuen Zertifikate erfolgen.

3.4.2. Zertifikatswechsel bei Änderung des Kommunikationsendpunktes

MUST Bei einem Zertifikatswechsel aufgrund einer Änderung des Kommunikationsendpunktes, z. B. bei neuen EDM-Systemen oder dem Wechsel eines Dienstleisters, MUSS der Marktpartner spätestens 10 Werktage vor Ungültigkeit der bisherigen Zertifikate die neuen Zertifikate zur Verfügung stellen. Somit entsteht ein Überlappungszeitraum von mindestens 10 Werktagen, in dem die bisherigen und die neuen Zertifikate gleichzeitig gültig sind.

MUST Innerhalb dieses Überlappungszeitraums MUSS der Marktpartner sicherstellen, dass:

- die Verzeichniseinträge im Verzeichnisdienst unverzüglich auf den neuen Kommunikationsendpunkt aktualisiert werden,
- die Verzeichniseinträge während der Übergangsphase konsistent und gültig bleiben,
- und Nachrichten, die der Data Hub gemäß dem jeweils gültigen Verzeichniseintrag an einen der während des Überlappungszeitraums gültigen Kommunikationsendpunkte zustellt, ordnungsgemäß verarbeitet werden können.

Durch diese Maßnahmen ist sicherzustellen, dass die Erreichbarkeit der API-Webdienste sowie die korrekte Auflösung der Kommunikationsparameter über den Verzeichnisdienst jederzeit gewährleistet bleibt.

3.5. Rückruf und Sperrlisten

MUST Will ein Zertifikatsinhaber sein Zertifikat vor Ablauf der Gültigkeitsfrist nicht mehr verwenden oder für ungültig erklären, so MUSS er sein Zertifikat über die Sperrlisten (CRL) seines CA-

Anbieters zurückziehen lassen. Die Vorgaben und Regelungen für die Sperrung von Zertifikaten, Verarbeitung von Sperrlisten und der Aktualisierungs- und Prüfungszeiten ergeben sich aus der Certificate Policy (CP) der SM-PKI [10].

MUST Wenn die CRL einer CA über den im Zertifikat eingetragenen Certificate Revocation List Distribution Point (CRL-DP) über 3 Tage nicht abrufbar ist oder im Gültigkeitszeitraum nicht verlängert wurde, MUSS der ausstellenden CA und allen darunter gelisteten Zertifikaten bis zur Veröffentlichung einer aktuellen CRL nach den Regelungen der CP misstraut werden.

4. Verwendung des Verzeichnisdienstes im Wasserstoffmarkt

Im Dokument „Regelungen zum Verzeichnisdienst“ des BDEW ist der Verzeichnisdienst für den Übertragungsweg API-Webdienste als Dienst zur Bereitstellung von Kommunikationsparametern beschrieben. Für den Wasserstoffmarkt wird dieses Prinzip unter Berücksichtigung der zentralen Data-Hub-Architektur konkretisiert.

Der Verzeichnisdienst wird im Wasserstoffmarkt zentral am Data Hub betrieben. Ein verteilter Betrieb mehrerer Verzeichnisdienste durch einzelne Marktpartner ist nicht vorgesehen.

Der Data Hub nutzt den Verzeichnisdienst als zentrale Metadaten-, Status- und Steuerungsgrundlage für die Verarbeitung fachlicher API-Kommunikation. Die im Verzeichnisdienst geführten Informationen dienen insbesondere der Prüfung, ob ein Marktpartner für einen fachlichen API-Webdienst in einer bestimmten Hauptversion zulässig, erreichbar, aktiv, berechtigt oder fachlich zuständig ist.

Der Verzeichnisdienst bildet damit die maßgebliche Grundlage für Routing-, Weiterleitungs-, Zustell-, Bereitstellungs- und Statusentscheidungen innerhalb der Data-Hub-Infrastruktur.

Die Verantwortung für die fachliche Aktualität marktpartnerbezogener Verzeichniseinträge verbleibt beim jeweiligen Marktpartner.

Die technischen Schnittstellen des Verzeichnisdienstes sind in den Kapiteln 4.6 und 4.7 (synchrone Web-API) sowie in Kapitel 5 (WebSocket-API) beschrieben.

4.1. Überblick Verzeichnisdienst

Der Verzeichnisdienst ist zentraler Bestandteil der Data-Hub-Infrastruktur und wird durch den Data Hub betrieben.

Der Verzeichnisdienst stellt die zentrale Datenbasis zur Verfügung, mit der der Data Hub und berechtigte Marktpartner feststellen können, welche Marktpartner welche fachlichen API-Webdienste in welcher Hauptversion unterstützen beziehungsweise für welche fachlichen API-Webdienste marktpartnerbezogene Kommunikationsparameter vorhanden sind.

Der Verzeichnisdienst enthält die für die Nutzung fachlicher API-Webdienste erforderlichen Kommunikationsparameter und Metadaten. Hierzu gehören insbesondere:

- Marktpartner-ID des verantwortlichen Marktpartners
- API-Kennung des fachlichen API-Webdienstes
- Hauptversion des fachlichen API-Webdienstes
- fachliche Bezeichnung des API-Webdienstes
- zulässige Marktrollen
- Kommunikationsparameter

- Capability-Informationen
- Lifecycle-Status
- Betriebsstatus
- technische und fachliche Kontaktinformationen
- Gültigkeitsinformationen
- Bereitstellungs- und Zustellinformationen innerhalb der Data-Hub-Infrastruktur
- Verfügbarkeitsüberwachung (`availabilityMonitoring`; marktpartnerseitig gepflegt)
- Status der Verfügbarkeitsüberwachung (`availabilityMonitoringStatus`; systemseitig geführt)
- Herkunft des Betriebsstatus und manueller Override (`operationalStatusSource`, `manualOverride`; systemseitig geführt)

Die Informationen im Verzeichnisdienst sind insbesondere dafür maßgeblich, ob ein Verzeichniseintrag für neue Routing-, Weiterleitungs-, Zustell- oder Bereitstellungsentscheidungen verwendet werden darf.

Die Verzeichniseinträge werden durch die Kombination aus Marktpartner-ID (MP-ID), API-Kennung und Hauptversion (`MAJOR`) eindeutig identifiziert. Dabei wird folgende Notation verwendet:

`<MP-ID>/<API-Kennung>/<MAJOR>`

Dabei gilt:

- **MP-ID** (*`providerId`*) bezeichnet die Marktpartner-ID des Marktpartners, für den der Verzeichniseintrag gilt.
- **API-Kennung** (*`apiId`*) bezeichnet die durch die API-Governance festgelegte Kennung des fachlichen API-Webdienstes.
- **`MAJOR`** (*`majorVersion`*) bezeichnet die Hauptversion des fachlichen API-Webdienstes.

Beispiel:

- `9766587654321/H2API00001/1`

Der Verzeichnisdienst unterstützt den Wechsel auf neue API-Versionen und Infrastrukturen durch Aktualisierung der Verzeichniseinträge. Er stellt hierfür die zentrale und maßgebliche Datenbasis dar.

4.2. API-Kennung

Jeder spezifizierte fachliche API-Webdienst erhält zur Bezeichnung innerhalb des Verzeichnisdienstes einen eigenen Identifikator. Dieser Identifikator wird als **API-Kennung** (`apiId`) bezeichnet.

Die API-Kennung identifiziert eindeutig den fachlichen API-Webdienst, auf den sich ein Verzeichniseintrag bezieht, und ist Bestandteil des eindeutigen Schlüssels eines Verzeichniseintrags. Die API-Kennung bezeichnet **nicht**:

- einen Marktpartner
- eine einzelne Nachricht
- eine Marktlokation
- eine Messlokation
- einen Vorgang
- ein Zertifikat
- eine Zieladresse
- eine technische URL

Die API-Kennung ist marktweit eindeutig und unabhängig von der technischen Umsetzung eines einzelnen Marktpartners. Eine Änderung der Zieladresse, des technischen Systems, eines Dienstleisters oder eines Zertifikats führt daher nicht zu einer neuen API-Kennung, sofern sich der fachliche API-Webdienst nicht ändert.

MUST Die API-Kennungen MÜSSEN durch die zuständige API-Governance in einer Anlage [12] zu dieser Guideline festgelegt und verwaltet werden.

Die Anlage enthält mindestens:

- API-Kennung (`apiId`)
- fachliche Bezeichnung des API-Webdienstes
- Kurzbeschreibung des fachlichen Zwecks
- zulässige Hauptversionen
- verantwortliche Governance-Stelle
- Datum der fachlichen Gültigkeit
- gegebenenfalls Nachfolger-API-Kennung oder Nachfolger-Version

MUST Die im Verzeichnisdienst verwendeten API-Kennungen MÜSSEN den in der Anlage festgelegten API-Kennungen entsprechen.

MUST NOT Marktpartner DÜRFEN KEINE eigenen API-Kennungen vergeben.

MUST NOT Marktpartner DÜRFEN KEINE Verzeichniseinträge mit API-Kennungen anlegen, die nicht durch die API-Governance festgelegt und im Verzeichnisdienst als zulässig gepflegt sind.

Beispiel für API-Kennungen in der Anlage:

- `H2API00001` = Übermittlung vorläufiger Messwerte
- `H2API00002` = Abfrage Verarbeitungsstatus vorläufiger Messwerte
- `H2API00003` = Stornierung vorläufiger Messwerte

MUST Für fachlich oder technisch eigenständige API-Webdienste MÜSSEN unterschiedliche API-Kennungen vergeben werden. Ob fachlich zusammengehörige Funktionen unter einer gemeinsamen API-Kennung oder unter unterschiedlichen API-Kennungen geführt werden, wird durch die API-Governance unter Berücksichtigung des fachlichen und technischen Schnittstellenzuschnitts festgelegt.

Eine `operationId` KANN innerhalb einer OpenAPI-Spezifikation zur eindeutigen technischen Kennzeichnung einzelner Operationen verwendet werden. Sie ist nicht Bestandteil des Routing- und Verzeichnismodells und hat keine Bedeutung für die Ermittlung des Verzeichniseintrags.

4.3. Verzeichniseintrag

Ein Verzeichniseintrag beschreibt die marktpartnerbezogenen Kommunikationsparameter zu einem fachlichen API-Webdienst in einer bestimmten Hauptversion.

Ein Verzeichniseintrag wird durch die Kombination aus MP-ID, API-Kennung und `MAJOR` eindeutig identifiziert.

Der Verzeichniseintrag enthält die für Routing, Weiterleitung, Zustellung, Bereitstellung oder technische Verarbeitung erforderlichen Kommunikationsparameter.

MUST Ein Verzeichniseintrag MUSS mindestens folgende Angaben enthalten:

- MP-ID
- API-Kennung
- `MAJOR`
- Zieladresse
- Kommunikationsprotokoll
- unterstützte HTTP-Methoden oder unterstützte Übertragungsmechanismen
- Lifecycle-Status
- Betriebsstatus
- Gültigkeitsbeginn
- technische Kontaktinformation
- Signatur des Verzeichniseintrags, soweit nach Schnittstellenspezifikation vorgesehen

Der Verzeichniseintrag kann sowohl durch den Marktpartner als auch durch die API-Governance oder den Data Hub gepflegte Bestandteile enthalten.

MUST Die jeweilige Datenhoheit und Änderungsberechtigung MUSS in der Schnittstellenspezifikation eindeutig festgelegt werden.

MUST NOT Marktpartner DÜRFEN durch die API-Governance oder den Data Hub gepflegte Bestandteile eines Verzeichniseintrags NICHT ändern.

MUST Die Signatur des Marktpartners MUSS ausschließlich die durch den Marktpartner zu pflegenden Bestandteile des Verzeichniseintrags umfassen.

MUST NOT Durch die API-Governance oder den Data Hub gepflegte Lifecycle-, Betriebsstatus- und sonstige systemseitige Informationen DÜRFEN NICHT Bestandteil der Signatur des Marktpartners sein. Dies gilt insbesondere für `operationalStatus`, `operationalStatusSource`, `manualOverride` und `availabilityMonitoringStatus`. Das marktpartnerseitig gepflegte Objekt `availabilityMonitoring` gehört dagegen zum signierten Teil des Verzeichniseintrags.

MUST NOT Automatische Änderungen des Betriebsstatus oder des `availabilityMonitoringStatus` durch den Data Hub DÜRFEN die Signatur der marktpartnerseitig gepflegten Bestandteile NICHT ungültig machen.

MUST Lifecycle-Status und Betriebsstatus MÜSSEN unterschiedliche Sachverhalte beschreiben und DÜRFEN NICHT gleichgesetzt werden.

MUST Der Lifecycle-Status MUSS die fachliche beziehungsweise strategische Einordnung des API-Webdienstes oder der API-Version beschreiben.

MUST Der Betriebsstatus MUSS die aktuelle technische Verfügbarkeit oder technische Einschränkung des konkreten marktpartnerbezogenen API-Webdienstes beschreiben.

MUST Der systemseitig geführte `availabilityMonitoringStatus` MUSS die aktuelle Aussagekraft der technischen Verfügbarkeitsüberwachung getrennt vom `operationalStatus` beschreiben und DARF NICHT mit diesem gleichgesetzt werden.

MUST Für `availabilityMonitoringStatus.state` sind ausschließlich die Werte `inactive`, `confirmed`, `unconfirmed` und `monitoringUnavailable` zulässig.

Die Zieladresse ist die technische Adresse, die der Data Hub zur weiteren Verarbeitung verwendet.

Die Zieladresse kann insbesondere bezeichnen:

- eine Routing-Adresse
- eine Weiterleitungsadresse
- eine Zustelladresse
- eine Bereitstellungsadresse
- einen technischen Übergabepunkt
- einen zentralen Data-Hub-Endpunkt
- einen marktpartnerbezogenen Endpunkt, soweit dieser für den jeweiligen API-Webdienst vorgesehen ist

Die Zieladresse ist nicht automatisch eine Adresse, die durch beliebige Clients direkt aufzurufen ist.

Die zulässige Nutzung einer Zieladresse ergibt sich aus:

- der jeweiligen API-Spezifikation
- dem Berechtigungskonzept
- dem Verzeichniseintrag

- den technischen Vorgaben des Data Hub

Für zentrale Data-Hub-Prozesse kann die Zieladresse auf eine Data-Hub-interne Verarbeitung oder auf einen zentralen Endpunkt verweisen.

Für marktpartnerbezogene Zustell- oder Bereitstellungsprozesse kann die Zieladresse auf einen vom Marktpartner gepflegten technischen Übergabepunkt verweisen, sofern dies für den jeweiligen API-Webdienst vorgesehen ist.

MUST Der Data Hub MUSS den Verzeichniseintrag zur Ermittlung der maßgeblichen Zieladresse und der zugehörigen Metadaten nutzen.

MAY Ein Verzeichniseintrag KANN zusätzlich Informationen zur Verwendbarkeit für neue Routing-, Weiterleitungs-, Zustell- oder Bereitstellungsentscheidungen enthalten.

Diese Verwendbarkeit kann insbesondere vom Lifecycle-Status, Betriebsstatus, Gültigkeitszeitraum, Berechtigungskontext und sonstigen prozessualen Vorgaben abhängen.

MUST Die Fehlerbehandlung für fehlende sowie syntaktisch oder formal ungültige Verzeichniseinträge MUSS gemäß Kapitel 4.6.3 erfolgen.

4.4. API-Aufruf

Der Data Hub nutzt den Verzeichnisdienst zur Ermittlung der für die Verarbeitung fachlicher API-Kommunikation erforderlichen Informationen.

Hierzu verwendet der Data Hub insbesondere den in Kapitel 4.2 definierten Schlüssel:

- `<MP-ID>/<API-Kennung>/<MAJOR>`

Bei fachlichen API-Aufrufen eines Marktpartners an den Data Hub ergibt sich der Schlüssel des Verzeichniseintrags aus den URL-Segmenten `<providerId>`, `<apiId>` und `<majorVersion>`.

Der Data Hub ermittelt anhand des zugehörigen Verzeichniseintrags die für die Verarbeitung erforderlichen Informationen gemäß Kapitel 4.3, insbesondere:

- Zieladresse
- Kommunikationsprotokoll
- unterstützte HTTP-Methoden oder Übertragungsmechanismen
- Betriebsstatus
- Lifecycle-Status
- Capability-Informationen
- technische Kontaktinformationen
- Gültigkeitsinformationen

MUST NOT Der Data Hub DARF die im Verzeichniseintrag enthaltene Zieladresse NICHT für andere Zwecke als die für den jeweiligen API-Webdienst vorgesehene Verarbeitung verwenden.

MUST Der Data Hub MUSS prüfen, ob der gefundene Verzeichniseintrag gültig und verwendbar ist.

Hierzu gehören mindestens folgende Prüfungen:

- Zulässigkeit der API-Kennung gemäß API-Katalog
- Zulässigkeit der `MAJOR`-Version
- Existenz eines Verzeichniseintrags
- Gültigkeit und Aktualität des Eintrags
- Zulässigkeit des Lifecycle-Status
- Zulässigkeit des Betriebsstatus
- Vorhandensein der Zieladresse
- Zulässigkeit der Zieladresse für den API-Webdienst
- Berechtigung des aufrufenden, adressierten oder fachlich betroffenen Marktpartners

MUST Bei fachlichen API-Aufrufen an den Data Hub MUSS der Wert des URL-Segments `<majorVersion>` mit dem für die adressierte Kombination aus `<providerId>` und `<apiId>` im Verzeichnisdienst geführten Wert `MAJOR` übereinstimmen.

MUST Ist eine dieser Prüfungen negativ, MUSS der Data Hub die weitere Verarbeitung ablehnen oder gemäß API-Spezifikation in einen definierten Fehlerprozess überführen.

MUST Der Data Hub MUSS Verzeichniseinträge mit dem Lifecycle-Status `retired` sowie Verzeichniseinträge mit dem Betriebsstatus `down` von Routing-, Weiterleitungs- und Zustellentscheidungen ausschließen, sofern die jeweilige Prozessbeschreibung für den Betriebsstatus `down` keine abweichende Fehler- oder Ersatzverarbeitung vorsieht. Die Behandlung der Betriebsstatus `maintenance` und `degraded` richtet sich nach den nachfolgenden Regelungen.

MUST NOT Ein Verzeichniseintrag mit einem Betriebsstatus `down` DARF grundsätzlich NICHT für neue Routing-, Weiterleitungs-, Zustell- oder Bereitstellungsentscheidungen verwendet werden, soweit die jeweilige Prozessbeschreibung keine abweichende Fehler- oder Ersatzverarbeitung vorsieht.

MAY Ein Verzeichniseintrag mit einem Betriebsstatus `maintenance` oder `degraded` KANN gemäß der jeweiligen Prozessbeschreibung weiterhin verwendet, eingeschränkt verwendet oder abgelehnt werden.

MUST Bei fachlichen API-Aufrufen eines Marktpartners an den Data Hub MUSS die API-Kennung als eigenes Segment des URL-Pfads übertragen werden.

MUST Der URL-Pfad fachlicher API-Aufrufe an den Data Hub MUSS dem Schema `<Basis-URL>/<providerId>/<apiId>/<majorVersion>` folgen. Die Segmente entsprechen dem Schlüssel des Verzeichniseintrags gemäß Kapitel 4.1: `<providerId>` bezeichnet die MP-ID des Marktpartners, für den der Verzeichniseintrag gilt (fachlicher Empfänger bzw. Anbieter des adressierten API-Webdienstes), `<apiId>` die API-Kennung und `<majorVersion>` die Hauptver-

sion ohne Präfix. Der Data Hub ermittelt anhand dieses Schlüssels den maßgeblichen Verzeichniseintrag und die Zieladresse. Für durch den Data Hub aufgerufene marktpartnerbezogene Endpunkte sowie für die Schnittstellen des Verzeichnisdienstes gilt diese Vorgabe nur, wenn die jeweilige Schnittstellenspezifikation dies ausdrücklich vorsieht.

MUST Der Data-Hub-Aufruf MUSS durch die Kombination aus `providerId`, `apiId`, `majorVersion` und HTTP-Methode einer konkreten Operation der zugehörigen API-Spezifikation eindeutig zugeordnet werden können. Umfasst eine API-Kennung mehrere fachlich zusammengehörige Funktionen, MÜSSEN diese ohne zusätzliche, in diesem Pfadschema nicht vorgesehene Pfadsegmente eindeutig unterscheidbar sein; andernfalls MÜSSEN unterschiedliche API-Kennungen vergeben werden.

MUST NOT Über den Verzeichnisdienst DÜRFEN KEINE Zertifikate verteilt oder abrufbar gemacht werden.

MUST Zertifikate MÜSSEN ausschließlich über die hierfür vorgesehenen Verzeichnisdienste der ausstellenden CA bezogen werden.

Die TLS-Authentisierung und die Signaturprüfung erfolgen nach den Vorgaben des jeweiligen Übertragungswegs und der jeweils geltenden Sicherheitsvorgaben.

Die fachlichen Inhaltsdaten werden entsprechend der jeweiligen API-Spezifikation übertragen.

Eine zusätzliche Verschlüsselung der Inhaltsdaten außerhalb der für den Übertragungsweg festgelegten Sicherheitsmechanismen ist nur erforderlich, wenn sie in der jeweiligen API-Spezifikation ausdrücklich vorgesehen ist.

4.5. Rollen und Verantwortlichkeiten

Für den Betrieb und die Nutzung des Verzeichnisdienstes im Wasserstoffmarkt bestehen folgende Rollen:

- Technischer Betreiber des Verzeichnisdienstes
- API-Governance
- Marktpartner

Der technische Betreiber des Verzeichnisdienstes ist verantwortlich für den Betrieb der Infrastruktur, über die der Verzeichnisdienst bereitgestellt wird.

Die API-Governance ist verantwortlich für die fachliche Pflege des API-Katalogs, insbesondere für die Vergabe und Verwaltung der API-Kennungen.

Marktpartner sind die fachlichen Akteure der Marktkommunikation. Sie können insbesondere als sendender Marktpartner, fachlich adressierter Marktpartner, zuständiger Marktpartner oder empfangsberechtigter Marktpartner auftreten.

Die Rolle API-Anbieter ist eine technische, kommunikationsstreckenbezogene Rolle. Eine direkte technische API-Kommunikation zwischen zwei Marktpartnern ist nicht vorgesehen.

Gegenüber einem aufrufenden Marktpartner ist der Data Hub API-Anbieter. Soweit der Data Hub einen marktpartnerbezogenen technischen Endpunkt aufruft, ist der Betreiber dieses Endpunkts API-Anbieter und der Data Hub API-Nutzer dieser Kommunikationsstrecke.

4.5.1. Pflichten des technischen Betreibers eines Verzeichnisdiensts

MUST Der technische Betreiber des Verzeichnisdienstes MUSS die für den sicheren, stabilen und ordnungsgemäßen Betrieb des Verzeichnisdienstes erforderlichen technischen Voraussetzungen schaffen und dauerhaft sicherstellen.

MUST Der technische Betreiber ist nicht für die fachliche Richtigkeit marktpartnerbezogener Verzeichniseinträge verantwortlich, soweit diese durch Marktpartner zu pflegen sind. Er MUSS jedoch sicherstellen, dass die hierfür vorgesehenen technischen Pflege-, Prüf-, Protokollierungs- und Benachrichtigungsmechanismen ordnungsgemäß bereitgestellt werden.

Für den technischen Betreiber des Verzeichnisdienstes gelten folgende Anforderungen:

MUST Der Zugang zum Verzeichnisdienst MUSS jedem berechtigten Teilnehmer der SM-PKI ermöglicht werden, der sich mit einem gültigen TLS-Client-Zertifikat der Klasse `EMT.API` authentisieren kann.

MUST NOT Eine Filterung ausschließlich auf Basis der IP-Adresse DARF NICHT erfolgen.

MUST NOT TLS-Client-Zertifikate aus anderen Zertifikatsklassen DÜRFEN NICHT als Ersatz für Zertifikate der Klasse `EMT.API` akzeptiert werden.

MUST Der Endpunkt des Verzeichnisdienstes MUSS über DNS adressierbar sein.

MUST Der Endpunkt des Verzeichnisdienstes MUSS über IPv4 erreichbar sein.

MAY Der Endpunkt des Verzeichnisdienstes DARF zusätzlich über IPv6 angeboten werden.

MUST Der Verzeichnisdienst MUSS hochverfügbar und ausfallsicher betrieben werden.

MUST Der technische Betreiber MUSS sicherstellen, dass berechtigte Marktpartner Verzeichnisinformationen abrufen können.

MUST Der technische Betreiber MUSS sicherstellen, dass berechtigte Marktpartner Verzeichnisinformationen einstellen, ändern oder löschen können, soweit die jeweilige Schnittstelle und das Berechtigungskonzept dies vorsehen.

MUST Der technische Betreiber MUSS die Integrität, Authentizität und Aktualität der über den Verzeichnisdienst bereitgestellten Informationen technisch unterstützen.

MUST Der technische Betreiber MUSS sicherstellen, dass Zugriffe auf den Verzeichnisdienst protokolliert werden, soweit dies für Betrieb, Nachvollziehbarkeit, Fehleranalyse und Sicherheit erforderlich ist.

MUST Der technische Betreiber MUSS geeignete Maßnahmen zur Erkennung, Behandlung und Dokumentation von Betriebsstörungen vorsehen.

MUST Soweit der Verzeichnisdienst automatische Statusmechanismen unterstützt, insbesondere zur Feststellung technischer Nichtverfügbarkeit, MUSS der technische Betreiber sicherstellen, dass diese Mechanismen nachvollziehbar, diskriminierungsfrei und ausschließlich für berechtigte Verzeichniseinträge angewendet werden.

MUST Der technische Betreiber MUSS sicherstellen, dass eine reine Abfrage oder eine reine Subscription auf einen Verzeichniseintrag nicht zu einer Änderung des Betriebsstatus dieses Verzeichniseintrags führt.

MUST Die Kontaktdaten eines technischen Ansprechpartners für den Verzeichnisdienst MÜSSEN stets aktuell gehalten und über die hierfür vorgesehene Serviceinformation bereitgestellt werden.

MUST Änderungen an technischen Serviceinformationen des Verzeichnisdienstes MÜSSEN den berechtigten Marktpartnern über die vorgesehenen Schnittstellen bereitgestellt werden.

SHOULD Wartungsfenster des Verzeichnisdienstes SOLLTEN mit dem in den geltenden Betriebsregelungen festgelegten Vorlauf angekündigt werden.

SHOULD Der technische Betreiber SOLLTE geeignete Monitoring- und Alarmierungsmechanismen einsetzen, um Störungen des Verzeichnisdienstes frühzeitig zu erkennen.

4.5.2. Pflichten der API-Governance

Die API-Governance ist verantwortlich für die Festlegung und Pflege der API-Kennungen.

Die Anlage der API-Kennungen ist die maßgebliche Quelle für alle im Verzeichnisdienst zulässigen API-Kennungen.

Die API-Governance ist insbesondere verantwortlich für:

- Vergabe von API-Kennungen
- Pflege der Anlage der API-Kennungen
- Freigabe von API-Kennungen
- Festlegung zulässiger Hauptversionen
- Festlegung fachlicher Bezeichnungen
- Festlegung von Nachfolger-API-Kennungen oder Nachfolger-Versionen
- Lifecycle-Steuerung auf Ebene des fachlichen API-Webdienstes
- Festlegung von Ablöse- und Migrationsregeln

Die API-Governance kann zulässige Lifecycle-Übergänge, Migrationspfade und Abweichungen von der regulären Lifecycle-Reihenfolge festlegen oder freigeben.

Abweichungen von der regulären Lifecycle-Reihenfolge DÜRFEN nur nach gesonderter Governance-Freigabe erfolgen.

Für die API-Governance gelten folgende Anforderungen:

MUST Die API-Governance MUSS die Eindeutigkeit von API-Kennungen sicherstellen.

MUST API-Kennungen MÜSSEN fortlaufende IDs besitzen und DÜRFEN NICHT wiederverwendet werden.

MUST Die API-Governance MUSS die zulässigen `MAJOR`-Versionen festlegen.

MUST Die API-Governance MUSS die zulässigen API-Kennungen im Verzeichnisdienst oder in der hierfür maßgeblichen Anlage pflegen.

MUST Änderungen an API-Kennungen und zulässigen `MAJOR`-Versionen MÜSSEN versioniert und für Audits nachvollziehbar dokumentiert werden.

MUST NOT Marktpartner DÜRFEN API-Kennungen NICHT eigenständig ändern.

4.5.3. Pflichten der Marktpartner

Marktpartner sind für die Pflege ihrer marktpartnerbezogenen Verzeichniseinträge verantwortlich.

Ein marktpartnerbezogener Verzeichniseintrag ist ein Eintrag, der durch die Kombination aus MP-ID, API-Kennung und `MAJOR` identifiziert wird und Kommunikationsparameter oder Metadaten dieses Marktpartners enthält.

Für Marktpartner gelten folgende Anforderungen:

MUST Ein Marktpartner MUSS für jeden fachlichen API-Webdienst und jede Hauptversion, für die er Kommunikationsparameter bereitstellungspflichtig ist, einen Verzeichniseintrag im zentralen Verzeichnisdienst pflegen.

MUST Ein Marktpartner MUSS seine Verzeichniseinträge stets aktuell halten.

MUST Änderungen an Zieladressen, Betriebsstatus, Kontaktinformationen oder sonstigen routingrelevanten Angaben MÜSSEN unverzüglich im Verzeichnisdienst gepflegt werden.

MUST Ein Marktpartner DARF nur solche Verzeichniseinträge anlegen, ändern oder löschen, die seiner eigenen MP-ID zugeordnet sind.

MUST Ein Marktpartner DARF nur solche API-Kennungen verwenden, die durch die API-Governance festgelegt und im Verzeichnisdienst als zulässig gepflegt sind.

MUST Ein Marktpartner MUSS die Zieladresse so pflegen, dass der Data Hub die vorgesehene Verarbeitung, Weiterleitung, Zustellung oder Bereitstellung durchführen kann.

MUST Ein Marktpartner MUSS sicherstellen, dass der im Verzeichnisdienst geführte Betriebsstatus den ihm bekannten technischen Zustand des betroffenen API-Webdienstes zutreffend wiedergibt.

MUST Soweit der Verzeichnisdienst eine technische Verfügbarkeitsmeldung über eine WebSocket-Verbindung unterstützt, MUSS der Marktpartner sicherstellen, dass diese nur für eigene Verzeichniseinträge und nur durch hierfür berechtigte Systeme verwendet wird.

MUST Ein Marktpartner MUSS sicherstellen, dass durch Aufbau, Abbruch oder Beendigung einer reinen Subscription kein Betriebsstatus eines Verzeichniseintrags verändert wird.

MUST Ein Marktpartner MUSS sicherstellen, dass die im Verzeichniseintrag enthaltene Zieladresse dem für den jeweiligen API-Webdienst zugelassenen Nutzungskontext entspricht.

MUST Ein Marktpartner MUSS sicherstellen, dass die von ihm gepflegten Verzeichniseinträge signiert sind, soweit die Schnittstellenspezifikation eine Signatur vorsieht.

MUST Die Signatur eines Verzeichniseintrags MUSS mit einem gültigen Signaturzertifikat der Klasse `EMT.API` des jeweiligen Marktpartners prüfbar sein, soweit die Schnittstellenspezifikation dies vorsieht.

MUST Bei Ablauf oder Sperrung eines Zertifikats MUSS die Signatur der betroffenen Verzeichniseinträge unverzüglich erneuert werden, soweit signierte Verzeichniseinträge verwendet werden.

MUST NOT Ein Marktpartner DARF NICHT Verzeichniseinträge für eine fremde MP-ID anlegen, ändern oder löschen.

MUST NOT Ein Marktpartner DARF NICHT eine Zieladresse pflegen, für deren Nutzung er nicht verantwortlich oder nicht berechtigt ist.

4.6. Nutzung/Abfrage von Informationen/Metadaten

Der Verzeichnisdienst stellt folgende Schnittstellen bereit:

- eine synchrone Web-API (Pull-Verfahren) zur gezielten Abfrage von Verzeichniseinträgen
- eine WebSocket-API (Push-Verfahren) zur Benachrichtigung über Änderungen an Verzeichniseinträgen; die Nutzung durch Clients ist optional

Über beide Schnittstellen werden konsistente Informationen bereitgestellt.

Die Nutzung der synchronen Web-API ist der maßgebliche Weg zur gezielten Ermittlung des aktuellen Verzeichniseintrags.

Die Nutzung der WebSocket-API dient der zeitnahen Information über Änderungen und ersetzt nicht die fachliche Prüfung, ob ein Verzeichniseintrag für einen konkreten Prozess verwendet werden darf.

4.6.1. Voraussetzungen

MUST Für die Nutzung des Verzeichnisdienstes MÜSSEN dem Client folgende Informationen bekannt sein:

- Endpunktadresse des Verzeichnisdienstes
- MP-ID

- API-Kennung
- MAJOR-Version

Zusätzlich MUSS der Client Teilnehmer der SM-PKI sein und über gültige Zertifikate der Klasse EMT.API verfügen.

Der Betriebsstatus des API-Webdienstes muss dem Client vor der Suche nicht bekannt sein. Der Betriebsstatus ist Bestandteil des Verzeichniseintrags und wird durch die Abfrage ermittelt.

4.6.2. Identifikation des Verzeichnisendpunkts

Der Verzeichnisdienst wird im Wasserstoffmarkt zentral bereitgestellt.

Die Endpunktadresse des zentralen Verzeichnisdienstes ist Bestandteil der technischen Rahmenvorgaben des Data Hub und wird den berechtigten Marktpartnern über die hierfür vorgesehenen Informationswege bereitgestellt.

Soweit die Vorgaben zur SM-PKI für den Wasserstoffmarkt die Angabe eines Verzeichnisdienst-Endpunkts in EMT.API-Zertifikaten vorsehen, ist hierfür die zentrale Endpunktadresse des Verzeichnisdienstes zu verwenden.

Die EMT.API-Zertifikate können über den LDAP-Verzeichnisdienst der ausstellenden Sub-CA der SM-PKI abgefragt werden.

Bei der Validierung der Gültigkeit von Zertifikaten sind die Vorgaben der CP der SM-PKI einzuhalten.

4.6.3. Gezielte Suche nach Verzeichniseinträgen

Der Verzeichnisdienst verfügt über eine Web-API, mit der gezielt nach einem Verzeichniseintrag gesucht werden kann.

Die Abfrage erfolgt über einen GET-Request per HTTPS an folgende URL:

- <Basis-URL>/v1/records/<MP-ID>/<API-Kennung>/<MAJOR>

Dabei ist:

- <Basis-URL> durch die Endpunktadresse des zentralen Verzeichnisdienstes zu ersetzen.
- <MP-ID> durch die H2-Marktpartner-ID des Marktpartners zu ersetzen, für den der Verzeichniseintrag gilt.
- <API-Kennung> durch die von der API-Governance festgelegte API-Kennung des fachlichen API-Webdienstes zu ersetzen.
- <MAJOR> durch die Hauptversion des fachlichen API-Webdienstes zu ersetzen.
- Das Pfadsegment v1 bezeichnet die Major-Version der Verzeichnisdienst-Schnittstelle gemäß Teil 1, Kapitel 4.2. <MAJOR> ist demgegenüber Bestandteil des Schlüssels des Verzeichniseintrags und wird ohne Präfix angegeben.

Beispiel:

- `GET https://vd.dh.de/v1/records/9871000000001/H2API00001/1`
- Bei Erfolg liefert der Verzeichnisdienst den entsprechenden Verzeichniseintrag gemäß Kapitel 4.3

MUST Der Verzeichnisdienst MUSS bedingte Leseanfragen unterstützen. Übermittelt der Client im Header `If-None-Match` den zuletzt empfangenen starken `ETag` und ist der Verzeichniseintrag unverändert, MUSS der Verzeichnisdienst mit `304 Not Modified` ohne Response-Body antworten; der Header `ETag` MUSS dabei den aktuellen Wert enthalten.

SHOULD Clients SOLLTEN bei der Resynchronisation nach einem Verbindungsaufbau bedingte `GET`-Anfragen mit `If-None-Match` verwenden, um unveränderte Verzeichniseinträge nicht vollständig zu übertragen ([WS-SPEC], Kapitel 9.3).

MUST NOT Eine Response mit `304 Not Modified` enthält keinen JSON-Body und wird daher NICHT auf Inhaltsdatenebene signiert (Teil 1, Kapitel 4.5.2).

MUST Ist die Kombination aus MP-ID, API-Kennung und `MAJOR` syntaktisch und formal gültig, jedoch kein entsprechender Verzeichniseintrag vorhanden, MUSS mit `404 Not Found` und einer `ErrorResponse` gemäß Teil 1, Kapitel 4.4.3 geantwortet werden.

MUST Enthält die Anfrage eine syntaktisch oder formal ungültige MP-ID, API-Kennung oder `MAJOR`-Version, MUSS mit `400 Bad Request` und einer `ErrorResponse` gemäß Teil 1, Kapitel 4.4.3 geantwortet werden.

4.6.4. Abonnieren von Verzeichniseinträgen

Der Verzeichnisdienst stellt eine WebSocket-API bereit, mit der Clients Änderungen an Verzeichniseinträgen abonnieren können.

Der Aufbau der WebSocket-Verbindung wird in Kapitel 5.2 beschrieben.

Eine Subscription bezieht sich auf einen Verzeichniseintrag, identifiziert durch:

- MP-ID
- API-Kennung
- `MAJOR`

4.7. Aktualisierung von Informationen/Metadaten

Die Aktualisierung von Verzeichnisdaten erfolgt über die hierfür vorgesehenen Schnittstellen und Prozesse des zentralen Verzeichnisdienstes.

Es wird zwischen folgenden Datenarten unterschieden:

- API-Katalogdaten
- marktpartnerbezogene Verzeichniseinträge
- Zieladressen

- Betriebsstatusdaten
- Lifecycle-Statusdaten
- Capability-Daten
- technische Kontaktinformationen
- Gültigkeitsinformationen

API-Katalogdaten werden durch die API-Governance gepflegt.

Marktpartnerbezogene Verzeichniseinträge werden durch den jeweils verantwortlichen Marktpartner gepflegt.

Statusdaten können abhängig von ihrer Art unterschiedlich entstehen:

- durch Pflege des verantwortlichen Marktpartners
- durch Pflege oder Freigabe der API-Governance
- durch technische Feststellung des Verzeichnisdienstes, soweit ein solcher Mechanismus vorgesehen ist

MUST Die Herkunft einer Statusänderung MUSS nachvollziehbar sein.

MAY Die Web-API des Verzeichnisdienstes KANN zusätzlich einen Mechanismus für partielle Änderungen bereitstellen, soweit dies in der Schnittstellenbeschreibung vorgesehen ist.

MUST Soweit dieser Mechanismus nicht bereitgestellt wird, MÜSSEN der technische Betreiber des Verzeichnisdienstes und die Marktpartner einen gleichwertigen Pflegeprozess verwenden. Dieser Pflegeprozess MUSS die Anforderungen an Authentisierung, Autorisierung, Aktualität, Nachvollziehbarkeit und Signatur erfüllen.

MUST Jeder aktive Verzeichniseintrag MUSS mit seiner aktuellen `revision` als starkem HTTP-Entity-Tag bereitgestellt werden. Der kanonische Wert MUSS `"rev-<revision>"` lauten und in `ETag` übertragen werden.

MUST Bei der erstmaligen Anlage eines Verzeichnisschlüssels, für den noch keine Revisions-Hochwassermarke besteht, MUSS `revision` den Wert 1 erhalten. Der zurückgegebene starke Entity-Tag MUSS entsprechend `"rev-1"` lauten. Wird ein zuvor gelöschter Verzeichniseintrag unter demselben Schlüssel erneut angelegt, MUSS `revision` um genau eins gegenüber der für diesen Schlüssel gespeicherten Revisions-Hochwassermarke erhöht werden. Ein Zurücksetzen auf 1 DARF NICHT erfolgen. Der zurückgegebene Entity-Tag MUSS `"rev-<revision>"` entsprechen.

MUST Lesezugriffe auf einen einzelnen Verzeichniseintrag MÜSSEN den aktuellen `ETag` zurückgeben. Änderungen mittels `PUT` oder `PATCH` sowie Löschungen mittels `DELETE` MÜSSEN den zuletzt gelesenen Wert im Header `If-Match` enthalten.

MUST Die initiale Anlage eines Eintrags MUSS den Header `If-None-Match: *` enthalten. Dadurch MUSS verhindert werden, dass ein zwischenzeitlich angelegter Eintrag unbeabsichtigt überschrieben wird.

MUST Fehlt eine vorgeschriebene Vorbedingung, MUSS der Verzeichnisdienst mit `428 Precondition Required` antworten. Entspricht `If-Match` nicht dem aktuellen `ETag` oder ist die Vorbedingung `If-None-Match: *` nicht erfüllt, MUSS er mit `412 Precondition Failed` antworten.

MUST Die Prüfung der Vorbedingung, die Berechtigungs- und Signaturprüfung, die Zustandsänderung, die Erhöhung der `revision` um genau eins, die Fortschreibung des Auditdatensatzes und die Anlage des zugehörigen Outbox-Ereignisses MÜSSEN in derselben atomaren Datenbanktransaktion erfolgen.

MUST Die vorstehende atomare Verarbeitung gilt für die erstmalige fachliche Ausführung. Bei einem identischen Retry gemäß Teil 1, Kapitel 4.5.3 MUSS das gespeicherte Ergebnis der ursprünglichen Anfrage ohne erneute Vorbedingungsprüfung, ohne Zustandsänderung, ohne Erhöhung der `revision` und ohne Erzeugung eines weiteren Outbox-Ereignisses zurückgegeben werden.

MUST Eine erfolgreich abgeschlossene Änderung DARF erst nach dem Commit für Abfragen und Benachrichtigungen sichtbar werden. Ein Fehler oder Rollback DARF weder eine erhöhte `revision` noch ein veröffentlichbares Ereignis hinterlassen.

MUST Benachrichtigungen MÜSSEN aus einer Transactional Outbox erzeugt werden. Jedes Outbox-Ereignis MUSS mindestens den Verzeichnisschlüssel, die neue `revision`, die Ereignisart und eine eindeutige Ereigniskennung enthalten. Für denselben Verzeichnisschlüssel MÜSSEN Ereignisse in aufsteigender Revisionsfolge verarbeitet werden.

MUST Die Übergabe eines Outbox-Ereignisses an aktive WebSocket-Verbindungen MUSS anhand von Verzeichnisschlüssel, `revision` und Verbindung idempotent erfolgen, sodass ein wiederholter Outbox-Verarbeitungsversuch keine mehrfache Zustandsänderung und innerhalb derselben Verbindung keine doppelte Benachrichtigung erzeugt.

Soweit dieser Mechanismus bereitgestellt wird, gelten die folgenden Unterkapitel.

4.7.1. Initiale Aufnahme von Einträgen

Die initiale Aufnahme eines neuen Verzeichniseintrags kann über einen PUT-Request per HTTPS erfolgen.

MUST Der Request MUSS `If-None-Match: *` enthalten.

Die URL lautet:

- `<Basis-URL>/v1/records/<MP-ID>/<API-Kennung>/<MAJOR>`

Dabei ist:

- `<Basis-URL>` durch die Endpunktadresse des zentralen Verzeichnisdienstes zu ersetzen.
- `<MP-ID>` durch die `H2-Marktpartner-ID` des Marktpartners zu ersetzen, für den der Eintrag gilt.

- `<API-Kennung>` durch die von der API-Governance festgelegte API-Kennung des fachlichen API-Webdienstes zu ersetzen.
- `<MAJOR>` durch die Hauptversion des fachlichen API-Webdienstes zu ersetzen.

Der HTTP-Body enthält die durch den Marktpartner zu pflegenden Bestandteile des anzulegenden Verzeichniseintrags.

MUST Die durch den Marktpartner zu pflegenden Bestandteile MÜSSEN vollständig sein und die in Kapitel 4.3 für den Marktpartner festgelegten Pflichtangaben enthalten.

MUST Vor der Anlage MUSS der Verzeichnisdienst mindestens prüfen:

- Ist der aufrufende Marktpartner authentisiert?
- Ist der aufrufende Marktpartner berechtigt, Einträge für die angegebene MP-ID anzulegen?
- Ist die API-Kennung durch die API-Governance zugelassen?
- Ist die angegebene `MAJOR`-Version für die API-Kennung zulässig?
- Ist die Zieladresse syntaktisch gültig?
- Sind die durch den Marktpartner zu pflegenden Bestandteile vollständig?
- Ist die Signatur gültig, soweit eine Signatur vorgesehen ist?

Bei Erfolg werden die übermittelten, durch den Marktpartner zu pflegenden Bestandteile in den Verzeichniseintrag übernommen. Der Verzeichniseintrag steht anschließend für nachfolgende Abfragen und Abonnements zur Verfügung.

MUST Soweit für den angelegten Verzeichniseintrag aktive Subscriptions bestehen oder allgemeine Benachrichtigungsmechanismen vorgesehen sind, MUSS der Verzeichnisdienst die Änderung gemäß Kapitel 5 bekannt machen.

4.7.2. Änderung von Einträgen

Die Änderung eines bestehenden Verzeichniseintrags kann über einen `PUT`-Request per HTTPS erfolgen.

MUST Der Request MUSS den zuletzt empfangenen starken `ETag` im Header `If-Match` enthalten.

Die URL lautet:

- `<Basis-URL>/v1/records/<MP-ID>/<API-Kennung>/<MAJOR>`

Der HTTP-Body enthält die vollständigen, durch den Marktpartner zu pflegenden Bestandteile des Verzeichniseintrags. Diese ersetzen die bisher durch den Marktpartner gepflegten Bestandteile.

MAY Soweit die Schnittstelle des Verzeichnisdienstes partielle Änderungen vorsieht, KANN eine Änderung einzelner Angaben auch über eine hierfür vorgesehene partielle Änderungsmethode oder über einen fachlich gleichwertigen Pflegeprozess erfolgen.

MUST Vor der Änderung MUSS der Verzeichnisdienst mindestens prüfen:

- Ist der aufrufende Marktpartner authentisiert?
- Ist der aufrufende Marktpartner berechtigt, den Eintrag für die angegebene MP-ID zu ändern?
- Ist die API-Kennung durch die API-Governance zugelassen?
- Ist die angegebene `MAJOR`-Version für die API-Kennung zulässig?
- Ist die Zieladresse syntaktisch gültig?
- Sind die durch den Marktpartner zu pflegenden Bestandteile vollständig?
- Ist die Signatur gültig, soweit eine Signatur vorgesehen ist?

Bei Erfolg ersetzen die übermittelten, durch den Marktpartner zu pflegenden Bestandteile die bisher durch den Marktpartner gepflegten Bestandteile des Verzeichniseintrags.

Nachfolgende Abfragen und Benachrichtigungen verwenden den aktualisierten Verzeichniseintrag.

MUST Änderungen an Zieladressen, Betriebsstatus oder anderen routingrelevanten Angaben MÜSSEN unverzüglich vorgenommen werden.

MUST Änderungen an einem Verzeichniseintrag MÜSSEN nachvollziehbar sein. Dies umfasst mindestens den Zeitpunkt der Änderung und die verantwortliche Stelle oder den verantwortlichen Marktpartner.

4.7.3. Löschung von Einträgen

Die Löschung eines bestehenden Verzeichniseintrags kann über einen `DELETE`-Request per HTTPS erfolgen.

MUST Der Request MUSS den zuletzt empfangenen starken `ETag` im Header `If-Match` enthalten.

Die URL lautet:

- `<Basis-URL>/v1/records/<MP-ID>/<API-Kennung>/<MAJOR>`

MUST Vor der Löschung MUSS der Verzeichnisdienst mindestens prüfen:

- Ist der aufrufende Marktpartner authentisiert?
- Ist der aufrufende Marktpartner berechtigt, den Eintrag für die angegebene MP-ID zu löschen?
- Bestehen fachliche, regulatorische oder technische Gründe, die einer Löschung entgegenstehen?

Bei Erfolg wird der Verzeichniseintrag aus dem operativen Verzeichnis gelöscht. Eine revisions- oder aufbewahrungsbedingte Archivierung außerhalb der operativen Nutzung bleibt hiervon unberührt.

MUST NOT Bestehen fachliche, regulatorische, technische oder revisionsbezogene Gründe, die einer Löschung aus dem operativen Verzeichnis entgegenstehen, DARF der `DELETE`-Request NICHT erfolgreich verarbeitet werden.

MUST Die Ablehnung MUSS mit HTTP `409 Conflict` und einer `ErrorResponse` gemäß Teil 1, Kapitel 4.4.3 erfolgen.

MUST Die fachliche Deaktivierung MUSS in diesem Fall über `PUT, PATCH` oder den hierfür vorgesehenen Pflegeprozess erfolgen.

MUST Eine dauerhafte fachliche Deaktivierung MUSS über den `lifecycleStatus` oder das Ende des Gültigkeitszeitraums abgebildet werden. Eine vorübergehende technische Nichtverfügbarkeit MUSS über den `operationalStatus` abgebildet werden.

Eine fachliche Deaktivierung ist gegenüber einer Löschung aus dem operativen Verzeichnis vorzuziehen, soweit der Verzeichniseintrag aus Gründen der Nachvollziehbarkeit, Revision, Migration oder regulatorischen Dokumentation weiterhin im Verzeichnis geführt werden muss.

MUST NOT Deaktivierte Verzeichniseinträge DÜRFEN NICHT mehr für neue Routing- oder Zustellentscheidungen verwendet werden, soweit sie nicht ausdrücklich für Übergangs- oder Migrationszwecke freigegeben sind.

MUST Die in Kapitel 4.7 festgelegten Vorbedingungen, Statuscodes und Signaturheader MÜSSEN in der maschinenlesbaren OpenAPI-Spezifikation des Verzeichnisdienstes gemäß Teil 1, Kapitel 4.4.1 abgebildet werden, damit generierte Clients die Header kennen, Implementierungen sie nicht auslassen und Konformitätstests die Vorbedingungen sowie die Statuscodes 412 und 428 prüfen können.

4.8. API-Lifecycle- und Betriebsstatus

Für marktpartnerbezogene API-Webdienste, die im Verzeichnisdienst des Data Hub geführt werden, gelten die folgenden Regeln hinsichtlich Lifecycle-Status und Betriebsstatus.

MUST Jeder Verzeichniseintrag MUSS einen Lifecycle-Status und einen Betriebsstatus enthalten.

MUST Lifecycle-Status und Betriebsstatus MÜSSEN getrennt betrachtet und gepflegt werden.

Der Lifecycle-Status beschreibt die fachliche beziehungsweise strategische Einordnung des API-Webdienstes oder der API-Version.

Der Betriebsstatus beschreibt die aktuelle technische Verfügbarkeit des konkreten marktpartnerbezogenen API-Webdienstes.

Änderungen am Lifecycle-Status und am Betriebsstatus können unterschiedliche Ursachen, Verantwortlichkeiten und Auswirkungen haben.

Alle Angaben zum API-Betriebsstatus sind Bestandteil des Verzeichniseintrags und unterliegen der Aktualisierungspflicht. Eine Signaturpflicht richtet sich nach der in der Schnittstellenspezifikation festgelegten Datenhoheit und dem dort festgelegten Signaturumfang.

4.8.1. Lifecycle-Status (fachlich / strategisch)

MUST Jede API-Version MUSS einen `lifecycleStatus` besitzen.

MUST Die Werte für `lifecycleStatus` MÜSSEN ausschließlich aus der folgenden Liste gewählt werden: `experimental`, `beta`, `active`, `deprecated`, `sunset`, `retired`.

MUST Die Angaben zum Lifecycle-Status MÜSSEN stets aktuell gehalten werden. Aktualisierungen bzw. Änderungen MÜSSEN unverzüglich durchgeführt werden.

MUST Bei Verwendung des `lifecycleStatus = deprecated` MUSS ein Datum `deprecatedSince` angegeben werden.

MUST Bei Verwendung des `lifecycleStatus = deprecated` MUSS ein eindeutiger Nachfolger (`replacement`) angegeben werden, sofern die API durch einen anderen API-Webdienst oder eine andere API-Version ersetzt wird. Erfolgt die Abkündigung ersatzlos, MUSS dies eindeutig kenntlich gemacht werden.

MUST Bei Verwendung des `lifecycleStatus = deprecated` MUSS ein Datum `sunsetAt` angegeben werden.

MUST Bei Verwendung des `lifecycleStatus = sunset` MUSS ein Datum `sunsetAt` angegeben werden.

MUST Das Datum `sunsetAt` MUSS zum Zeitpunkt der Setzung in der Zukunft liegen.

MUST Eine API im Status `sunset` MUSS weiterhin technisch erreichbar sein, bis das Sunset-Datum erreicht ist.

MUST Eine API im Status `retired` MUSS technisch deaktiviert werden, z. B. durch Rückgabe des HTTP-Statuscodes `410 Gone` oder durch eine entsprechende Sperrung im API-Gateway.

MUST Statusübergänge des `lifecycleStatus` MÜSSEN grundsätzlich in aufsteigender Reihenfolge `experimental` → `beta` → `active` → `deprecated` → `sunset` → `retired` erfolgen.

MAY Zwischenstatus KÖNNEN übersprungen werden.

MUST Rückschritte oder sonstige Abweichungen von dieser Reihenfolge DÜRFEN ausschließlich nach gesonderter, nachvollziehbar dokumentierter Freigabe der API-Governance erfolgen.

MUST NOT Ein `lifecycleStatus = deprecated` DARF NICHT ohne Angabe von `deprecatedSince` und `sunsetAt` verwendet werden. Sofern ein Nachfolger besteht, MUSS zusätzlich `replacement` angegeben werden.

MUST NOT Ein `lifecycleStatus = sunset` DARF NICHT ohne Angabe von `sunsetAt` verwendet werden.

4.8.2. Betriebsstatus (technisch)

MUST Jeder marktpartnerbezogene Verzeichniseintrag MUSS einen `operationalStatus` besitzen.

MUST Die Werte für `operationalStatus` MÜSSEN ausschließlich aus der folgenden Liste gewählt werden: `operational`, `degraded`, `maintenance`, `down`.

MUST Der `operationalStatus` MUSS getrennt vom `lifecycleStatus` verwaltet werden.

MUST Bei Verwendung des `operationalStatus` = `maintenance` MUSS ein Wartungszeitraum (`maintenanceWindow`) angegeben werden.

MUST Der Wartungszeitraum MUSS ein Start- und Enddatum enthalten.

MUST Bei Verwendung des `operationalStatus = degraded` MUSS die Art der Einschränkung dokumentiert sein.

MUST Bei `lifecycleStatus = retired` MUSS der `operationalStatus` den Wert `down` besitzen.

MUST NOT Ein `operationalStatus = maintenance` DARF NICHT ohne Wartungszeitraum verwendet werden.

MUST NOT Eine API im Status `retired` DARF NICHT weiterhin erreichbar sein.

Der Betriebsstatus kann durch den verantwortlichen Marktpartner gepflegt oder, soweit die Schnittstelle des Verzeichnisdienstes dies vorsieht, durch technische Feststellung des Verzeichnisdienstes aktualisiert werden.

MUST Die Herkunft, der Zeitpunkt und der auslösende Mechanismus einer Änderung des Betriebsstatus MÜSSEN nachvollziehbar sein.

MUST Soweit der Verzeichnisdienst einen Betriebsstatus automatisch setzt, MUSS die technische Schnittstellenbeschreibung des Verzeichnisdienstes festlegen, unter welchen Voraussetzungen der automatisch gesetzte Status durch den Marktpartner geändert oder aufgehoben werden darf.

Eine technische Feststellung des Verzeichnisdienstes darf insbesondere auf Grundlage einer hierfür vorgesehenen technischen Verfügbarkeitsmeldung erfolgen.

4.8.3. Tabellarische Kurzreferenz

4.8.3.1. Lifecycle-Status

Status	Bedeutung	Pflichtfelder
`experimental`	Frühe Phase / Testbetrieb	—

Status	Bedeutung	Pflichtfelder
`beta`	Eingeschränkt stabil	—
`active`	Produktivbetrieb	—
`deprecated`	Abgekündigt	`deprecatedSince, sunsetAt; replacement bei Ablösung bzw. Kennzeichnung „ersatzlos“`
`sunset`	Abschaltung geplant	`sunsetAt`
`retired`	Abgeschaltet	—

4.8.3.2. Betriebsstatus

Status	Bedeutung	Zusatzanforderungen
`operational`	Voll verfügbar	—
`degraded`	Eingeschränkt	Beschreibung der Einschränkung
`maintenance`	Wartung	`maintenanceWindow`
`down`	Nicht verfügbar	Incident / Monitoring erforderlich

4.8.4. Automatische Aktualisierung des Betriebsstatus

MAY Der Verzeichnisdienst KANN Mechanismen unterstützen, mit denen die technische Verfügbarkeit eines marktpartnerbezogenen API-Webdienstes festgestellt oder plausibilisiert wird.

MUST Eine automatische Aktualisierung des Betriebsstatus DARF NUR erfolgen, wenn hierfür ein ausdrücklich vorgesehener Mechanismus verwendet wird und der betroffene Marktpartner für den jeweiligen Verzeichniseintrag berechtigt ist.

MUST NOT Eine reine Abfrage, eine reine WebSocket-Verbindung oder eine reine Subscription auf einen Verzeichniseintrag DARF NICHT zu einer automatischen Änderung des Betriebsstatus führen.

MUST Der Verzeichnisdienst MUSS die Aussagekraft Lifecycle-Status getrennt vom Betriebsstatus im systemseitigen `availabilityMonitoringStatus` führen.

MUST Nur ein endpunktspezifischer Wegfall einer Verfügbarkeitsmeldung DARF nach den in der technischen Schnittstellenbeschreibung festgelegten Fristen zu einer automatischen Änderung des `operationalStatus` auf `down` führen.

MUST Bei einer Störung der Überwachungsinfrastruktur MUSS `availabilityMonitoringStatus.state` auf `monitoringUnavailable` gesetzt werden. Der `operationalStatus` DARF allein aufgrund dieser Störung NICHT geändert werden.

MUST Der Verzeichnisdienst MUSS einen clusterweit wirksamen Mass-Loss-Circuit-Breaker vorsehen, der automatische `down`-Übergänge bei ungewöhnlich vielen zeitlich oder technisch korrelierten Wegfällen unterbindet.

MUST Nach Wiederherstellung der Überwachungsinfrastruktur DÜRFEN während der Störung abgelaufene Heartbeat-, Timeout- oder Karenzfristen NICHT rückwirkend zu einer automatischen Statusänderung führen. Eine Neubewertung MUSS mit neu beginnenden Fristen erfolgen.

Soweit der Verzeichnisdienst eine technische Verfügbarkeitsmeldung über WebSocket unterstützt, gelten die detaillierten Regelungen in Kapitel 5.

5. WebSocket-Schnittstelle des Verzeichnisdienst

5.1. Überblick

Der Verzeichnisdienst stellt neben der synchronen Web-API eine asynchrone WebSocket-API bereit, über die Clients Änderungen an Verzeichniseinträgen zeitnah empfangen können.

Die WebSocket-API dient der ereignisbasierten Bereitstellung von Informationen und ermöglicht es, Änderungen an Verzeichniseinträgen automatisiert und ohne wiederholte synchrone Abfragen zu verarbeiten.

Die Nutzung der WebSocket-API ist optional und erfolgt ergänzend zur synchronen Abfrage von Verzeichniseinträgen über die Web-API. Beide Schnittstellen stellen konsistente Informationen bereit.

Die WebSocket-API wird ausschließlich über den zentralen Verzeichnisdienst am Data Hub bereitgestellt und unterstützt die effiziente Synchronisation von Metadaten zwischen dem Verzeichnisdienst und den Systemen der Marktpartner.

Die WebSocket-API kann zwei unterschiedliche Zwecke unterstützen:

- Subscriptions zum Empfang von Änderungen an Verzeichniseinträgen
- technische Verfügbarkeitsmeldungen für eigene Verzeichniseinträge, soweit diese Funktion vorgesehen ist

Subscriptions und technische Verfügbarkeitsmeldungen sind fachlich getrennt zu betrachten.

Eine Subscription dient dem Empfang von Informationen.

Eine technische Verfügbarkeitsmeldung dient der Anzeige oder Feststellung, dass ein bestimmter marktpartnerbezogener API-Webdienst technisch verfügbar ist beziehungsweise durch das verantwortliche System als verfügbar gemeldet wird.

MUST Für die über die WebSocket-Schnittstelle ausgetauschten JSON-Nachrichten gelten die Anforderungen aus Teil 1, Kapitel 4.3 — insbesondere UTF-8-Kodierung ohne BOM, I-JSON-Konformität und die Validierbarkeit gegen das jeweils gültige Nachrichtenschema — entsprechend.

MUST Die konkrete technische Ausgestaltung der WebSocket-API MUSS in einer gesonderten technischen Schnittstellenbeschreibung festgelegt werden.

MUST Die technische Schnittstellenbeschreibung MUSS mindestens den Verbindungsendpunkt und die Schnittstellenversion, die Nachrichtentypen und Nachrichtenschemata, Korrelations- und Revisionskennungen, die Verwaltung von Subscriptions, Fehlernachrichten und Close-Codes, das Zustell- und Wiederanlaufverhalten sowie Parameter für Heartbeat, Timeout und maximale Nachrichtengröße festlegen.

5.2. Aufbau der WebSocket-Verbindung

Der Client baut über die folgende URI eine TLS-gesicherte WebSocket-Verbindung zum zentralen Verzeichnisdienst auf:

- `wss://<Host>/v1/ws/subscriptions`

Dabei ist:

- `<Host>` durch den DNS-Namen des zentralen Verzeichnisdiensts am Data Hub zu ersetzen.

MUST Die WebSocket-Verbindung MUSS ausschließlich über `WSS` und als mTLS-Verbindung gemäß Kapitel 2.2 aufgebaut werden.

MUST Der Client MUSS beim Verbindungsaufbau das gültige `TLS`-Client-Zertifikat seines `EMT.API`-Zertifikatstripels verwenden und die Anforderungen der BSI TR-03116-3 einhalten.

Der Verbindungsaufbau erfolgt mittels HTTP-Upgrade über TLS. Bei erfolgreichem Verbindungsaufbau wird die Verbindung für den asynchronen Nachrichtenaustausch genutzt.

MUST Der Verzeichnisdienst MUSS den Marktpartner aus der Authentisierung eindeutig bestimmen können, soweit über die WebSocket-Verbindung Subscriptions oder technische Verfügbarkeitsmeldungen verwaltet werden.

MUST Der durch die mTLS-Authentisierung hergestellte Sicherheitskontext MUSS für die Dauer der jeweiligen WebSocket-Verbindung gelten. Sämtliche über die Verbindung verwalteten Subscriptions und technischen Verfügbarkeitsmeldungen MÜSSEN dem authentisierten Marktpartner zugeordnet und entsprechend dessen Berechtigungen verarbeitet werden.

Der HTTP-Request zum Aufbau der WebSocket-Verbindung wird nicht auf Inhaltsdatenebene signiert. Die Authentisierung und die Zuordnung des Marktpartners erfolgen über die mTLS-Verbindung gemäß Kapitel 2.2. Das `SIG`-Zertifikat des verwendeten `EMT.API`-Zertifikatstripels MUSS jedoch gemäß technischer Schnittstellenbeschreibung im Upgrade-Request übertragen, geprüft und für die Dauer der Verbindung an den Sicherheitskontext gebunden werden.

MUST Nachrichten innerhalb der WebSocket-Verbindung, die Subscriptions einrichten oder kündigen oder die als technische Verfügbarkeitsmeldung dienen, MÜSSEN mit dem privaten Schlüssel des beim Verbindungsaufbau gebundenen `SIG`-Zertifikats auf Nachrichtenebene signiert werden. Für die Signaturbildung gelten die Vorgaben aus Kapitel 2.3 entsprechend; die JSON-Nachricht MUSS vor der Signaturberechnung gemäß RFC 8785 kanonisiert werden. Die konkrete Signaturübertragung MUSS in der technischen Schnittstellenbeschreibung festgelegt werden.

MUST Jede signaturpflichtige Client-Nachricht MUSS kryptographisch an den konkreten Verbindungsaufbau gebunden werden. Hierzu MUSS die `H2-Transaction-Id` des erfolgreichen HTTP-Upgrade-Requests als `connectionId` in die Signaturrepräsentation einbezogen werden; sie wird nicht als zusätzliche Eigenschaft im WebSocket-Envelope übertragen.

MUST Jeder neue Verbindungsaufbau MUSS eine neue `connectionId` verwenden. Signaturen einer beendeten oder einer anderen WebSocket-Verbindung DÜRFEN in der aktuellen Verbindung NICHT gültig sein.

MAY Nachrichten des Verzeichnisdienstes an Clients (insbesondere Benachrichtigungen, Bestätigungs- und Fehlernachrichten) KÖNNEN unsigniert übertragen werden, soweit die technische Schnittstellenbeschreibung nichts Abweichendes festlegt.

MUST Die technische Schnittstellenbeschreibung und die JWS-Spezifikation MÜSSEN die Bildung, Übertragung und Prüfung der Nachrichtensignatur innerhalb der WebSocket-Verbindung einschließlich der Bindung an die `connectionId` verbindlich festlegen. Die in der technischen Schnittstellenbeschreibung bezeichneten Client-Nachrichten MÜSSEN entsprechend diesen Vorgaben signiert werden.

Technische WebSocket-Ping-/Pong-Mechanismen dienen ausschließlich der Erkennung und Aufrechterhaltung der Verbindung. Sie begründen für sich genommen keine fachliche Aussage über die Verfügbarkeit eines API-Webdienstes.

5.3. Subscriptions

5.3.1. Allgemeines

Die WebSocket-API basiert auf dem Konzept sogenannter Subscriptions.

Ein Client KANN gezielt Verzeichniseinträge abonnieren, um über deren Änderungen informiert zu werden. Eine Subscription bezieht sich dabei eindeutig auf einen Verzeichniseintrag, der durch folgende Parameter identifiziert wird:

- Marktpartner-ID (*`providerId`*),
- API-Kennung (*`apiId`*),
- Hauptversion (*`majorVersion`*).

Eine Subscription ist eine Beobachtung eines Verzeichniseintrags. Sie ist keine technische Verfügbarkeitsmeldung des betroffenen API-Webdienstes.

MUST NOT Der Abbruch, die Beendigung oder der Verlust einer reinen Subscription DARF den Betriebsstatus des betroffenen Verzeichniseintrags NICHT verändern.

5.3.2. Einrichtung und Verwaltung

Der Client verwaltet Subscriptions durch das Senden von Nachrichten an den Verzeichnisdienst.

Eine Subscription:

- MUSS durch den Server bestätigt werden, bevor sie als aktiv gilt,
- gilt für die Dauer der WebSocket-Verbindung oder bis zu ihrer Kündigung,
- KANN durch den Client jederzeit gekündigt werden.

MUST Der Server MUSS:

- eingehende Subscription-Anfragen verarbeiten,
- den Client über das Ergebnis informieren.

MUST Der Server MUSS unzulässige oder nicht berechtigte Subscription-Anfragen ablehnen.

MUST Der Verzeichnisdienst DARF einem Client nur solche Informationen übermitteln, für deren Empfang der Client berechtigt ist.

5.3.3. Verhalten bei mehrfachen Anfragen

MUST Die Verwaltung von Subscriptions MUSS idempotent erfolgen.

Das bedeutet:

- Mehrfache Anforderungen derselben Subscription führen nicht zu mehrfachen Einträgen.
- Die Kündigung einer nicht existierenden Subscription DARF NICHT als Fehler behandelt werden.

5.4. Benachrichtigungen

5.4.1. Allgemeines

MUST Der Verzeichnisdienst MUSS Clients über Änderungen an Verzeichniseinträgen informieren, sofern eine aktive Subscription besteht.

Benachrichtigungen erfolgen:

- als Antwort auf Subscription-Anfragen,
- sowie ereignisbasiert bei Änderungen im Verzeichnis.

Eine Benachrichtigung über eine Änderung ersetzt nicht die Pflicht des Clients, die Verwendbarkeit eines Verzeichniseintrags für den konkreten Prozess zu prüfen.

5.4.2. Arten von Benachrichtigungen

Der Verzeichnisdienst unterscheidet folgende Ereignisse:

a) Hinzufügen oder Aktualisierung

Neue oder geänderte Verzeichniseinträge werden an berechtigte abonnierte Clients gemeldet.

b) Löschung

Endgültig aus dem operativen Verzeichnis gelöschte Verzeichniseinträge werden an berechtigte abonnierte Clients gemeldet. Die Deaktivierung eines Eintrags ist keine Löschung; deaktivierte Verzeichniseinträge bleiben Bestandteil des Verzeichnisses.

c) Änderung von Statusinformationen

Änderungen am Lifecycle-Status oder Betriebsstatus werden an berechtigte abonnierte Clients gemeldet. Dies umfasst auch die fachliche oder technische Deaktivierung eines weiterhin im Verzeichnis geführten Eintrags.

d) Änderung von Service-Informationen

Technische Änderungen, zum Beispiel Version oder Kontakt, werden als Serviceinformation bereitgestellt.

e) Beendigung von Subscriptions

Gekündigte, abgelehnte oder serverseitig beendete Subscriptions werden dem betroffenen Client mitgeteilt.

Benachrichtigungen können den geänderten Verzeichniseintrag vollständig, auszugsweise oder referenzierend enthalten. Der genaue Umfang wird in der jeweiligen Schnittstellenbeschreibung festgelegt.

5.4.3. Umfang der Benachrichtigungen

MUST Der Server MUSS sicherstellen, dass er:

- nur über Einträge informiert, für die eine aktive Subscription besteht
- nur solche Informationen übermittelt, für die der Client berechtigt ist

MUST Der Server MUSS:

- alle relevanten Änderungen an abonnierte Clients übermitteln
- die Benachrichtigung zeitnah nach Eintritt eines Ereignisses versenden
- sicherstellen, dass Benachrichtigungen für Clients nachvollziehbar zuordenbar sind

SHOULD Clients SOLLTEN empfangene Benachrichtigungen so verarbeiten, dass mehrfache oder verspätete Benachrichtigungen nicht zu einem fehlerhaften lokalen Zustand führen.

SHOULD Soweit Revisionsstände oder vergleichbare Änderungskennzeichen verwendet werden, SOLLTEN Clients diese berücksichtigen.

5.5. Fehlerbehandlung

MUST Treten bei der Verarbeitung von Subscription-Anfragen oder bei der Verarbeitung von Ereignissen Fehler auf, MUSS der Verzeichnisdienst den Client informieren.

MUST Die Fehlerbenachrichtigung MUSS strukturierte Informationen über den Fehler enthalten.

MUST Die Fehlerbenachrichtigung MUSS erkennen lassen, auf welche Anfrage oder welchen Sachverhalt sich der Fehler bezieht.

MUST NOT Die Fehlerbenachrichtigung DARF den Zustand bestehender Subscriptions NICHT verändern.

MAY Im Falle kritischer Fehler, bei denen der korrekte Betrieb nicht mehr gewährleistet werden kann, DARF die WebSocket-Verbindung durch den Server beendet werden.

MUST NOT Eine fehlerhafte oder nicht berechtigte technische Verfügbarkeitsmeldung DARF NICHT dazu führen, dass der Betriebsstatus eines Verzeichniseintrags unberechtigt geändert wird.

5.6. Anforderungen an Clients

MUST Clients, die die WebSocket-API nutzen, MÜSSEN:

- aktive Subscriptions verwalten können,
- empfangene Benachrichtigungen verarbeiten,
- Änderungen an Verzeichniseinträgen zeitnah berücksichtigen,
- mit Verbindungsabbrüchen umgehen und Subscriptions bei Bedarf erneut aufbauen.

SHOULD Clients SOLLTEN:

- bekannte Revisionsstände berücksichtigen,
- unnötige Subscriptions vermeiden.

SHOULD Clients SOLLTEN nach Verbindungsabbrüchen den aktuellen Stand relevanter Verzeichniseinträge über die synchrone Web-API prüfen, soweit eine lückenlose Verarbeitung nicht sichergestellt werden kann.

MUST Clients, die technische Verfügbarkeitsmeldungen verwenden, MÜSSEN sicherstellen, dass diese nur für eigene und berechtigte Verzeichniseinträge erfolgen.

5.7. Einordnung in die Marktkommunikation

Die WebSocket-API ermöglicht eine effiziente und automatisierte Synchronisation von Verzeichniseinträgen zwischen Verzeichnisdienst und den Systemen der Marktpartner.

Durch die Kombination aus:

- synchroner Web-API (Pull) und
- asynchroner WebSocket-API (Push)

wird ein robuster und skalierbarer Betrieb der Marktkommunikation im Wasserstoffmarkt unterstützt.

Die WebSocket-API ist Bestandteil des zentralen Verzeichnisdiensts am Data Hub und wird ausschließlich dort bereitgestellt.

Die WebSocket-API begründet keine direkte technische Marktkommunikation zwischen Marktpartnern. Sie dient ausschließlich der Kommunikation zwischen berechtigten Clients und dem zentralen Verzeichnisdienst am Data Hub.

5.8. Technische Verfügbarkeitsmeldung

MAY Der Verzeichnisdienst KANN eine technische Verfügbarkeitsmeldung über WebSocket unterstützen.

Eine technische Verfügbarkeitsmeldung dient dazu, dem Verzeichnisdienst mitzuteilen, dass ein bestimmter marktpartnerbezogener API-Webdienst durch das verantwortliche System des Marktpartners als technisch verfügbar gemeldet wird.

Eine technische Verfügbarkeitsmeldung ist von einer Subscription zu unterscheiden. Eine Subscription dient dem Empfang von Änderungen; eine technische Verfügbarkeitsmeldung dient der Anzeige oder Feststellung technischer Verfügbarkeit.

MUST Eine technische Verfügbarkeitsmeldung DARF nur für Verzeichniseinträge erfolgen, für die der authentisierte Marktpartner verantwortlich und berechtigt ist.

MUST NOT Eine technische Verfügbarkeitsmeldung DARF NICHT für Verzeichniseinträge fremder MP-IDs erfolgen.

MUST NOT Eine reine Subscription DARF NICHT als technische Verfügbarkeitsmeldung behandelt werden.

MUST Der Verzeichnisdienst MUSS je überwachten Verzeichniseintrag das folgende systemseitige Objekt `availabilityMonitoringStatus` führen:

```
"availabilityMonitoringStatus": {
  "state": "confirmed",
  "changedAt": "2026-07-12T09:31:01Z",
  "reason": "availabilityConfirmed"
}
```

MUST `availabilityMonitoringStatus` DARF NICHT durch den Marktpartner geändert werden und DARF NICHT Bestandteil der Marktpartnersignatur des Verzeichniseintrags sein.

MUST `state` MUSS einen der Werte `inactive`, `confirmed`, `unconfirmed` oder `monitoringUnavailable` enthalten. `changedAt` MUSS den Zeitpunkt des letzten Zustandswechsels in UTC enthalten; `reason` MUSS einen maschinenlesbaren `lowerCamelCase`-Code enthalten. Ist `availabilityMonitoring.enabled = true`, liegt jedoch noch keine bestätigte technische Verfügbarkeitsmeldung vor, MUSS `state` den Wert `unconfirmed` besitzen. In diesem Zustand DARF allein aufgrund des Ausbleibens der erstmaligen Verfügbarkeitsmeldung keine Karenzzeit für einen automatischen Übergang des `operationalStatus` auf `down` beginnen. `inactive`

DARF nur verwendet werden, wenn die Überwachung deaktiviert ist oder `lifecycleStatus = retired` gilt.

MUST Eine Änderung von `availabilityMonitoringStatus.state` MUSS die Revision des Verzeichniseintrags erhöhen. Ein Heartbeat, der den bereits bestehenden Zustand `confirmed` lediglich verlängert, DARF keine Revision erzeugen.

5.8.1. Heartbeat bei technischer Verfügbarkeitsmeldung

MAY Soweit der Verzeichnisdienst technische Verfügbarkeitsmeldungen unterstützt, KANN hierfür ein Heartbeat-Mechanismus vorgesehen werden.

Der fachliche Heartbeat wird vom berechtigten Client beziehungsweise vom technischen System des verantwortlichen Marktpartners an den Verzeichnisdienst gesendet.

MAY Der Verzeichnisdienst KANN den Empfang des Heartbeats bestätigen.

MUST Soweit ein Heartbeat-Mechanismus verwendet wird, MÜSSEN Heartbeat-Intervall, Timeout und eine gegebenenfalls anzuwendende Karenzzeit in der technischen Schnittstellenbeschreibung verbindlich festgelegt werden.

Technische WebSocket-Ping-/Pong-Frames sind hiervon zu unterscheiden. Sie dienen ausschließlich der technischen Verbindungserkennung und ersetzen keinen fachlichen Heartbeat für eine Verfügbarkeitsmeldung.

5.8.2. Wegfall der technischen Verfügbarkeitsmeldung

MUST Soweit eine aktive technische Verfügbarkeitsmeldung besteht, MUSS der Verzeichnisdienst den Wegfall dieser Verfügbarkeitsmeldung erkennen können.

MUST NOT Das bloße Ausbleiben einer erstmaligen technischen Verfügbarkeitsmeldung nach Aktivierung der Überwachung DARF ohne vorherige bestätigte Verfügbarkeitsmeldung nicht als Wegfall gewertet werden.

Ein Wegfall kann insbesondere vorliegen bei:

- ordnungsgemäßer Beendigung der Verfügbarkeitsmeldung durch den Client,
- Abbruch der WebSocket-Verbindung,
- Ausbleiben eines vorgesehenen Heartbeats oder
- serverseitiger Beendigung der Verbindung.

MUST Die technische Schnittstellenbeschreibung MUSS bei einer ordnungsgemäßen Beendigung einen maschinenlesbaren Grund vorsehen und zwischen einer ausdrücklich gemeldeten Nichtverfügbarkeit des API-Webdienstes und der bloßen Beendigung oder Verlagerung des Monitoring-Clients unterscheiden.

MUST Der Verzeichnisdienst MUSS beim Wegfall zwischen einem endpunktspezifischen Wegfall und einer Störung der Überwachungsinfrastruktur unterscheiden.

Als endpunktspezifischer Wegfall gelten insbesondere eine ausdrücklich gemeldete Nichtverfügbarkeit des API-Webdienstes, ein isoliertes Heartbeat-Timeout oder ein isolierter Verlust einer Clientverbindung, sofern die Überwachungsinfrastruktur ordnungsgemäß arbeitet und keine zeitlich oder technisch korrelierte Störung vorliegt.

Als Störung der Überwachungsinfrastruktur gelten insbesondere eine geplante oder fehlerbedingte serverseitige Beendigung, eine Störung des WebSocket-Dienstes, seiner Datenhaltung, des Clusters, der PKI-, DNS- oder Netzabhängigkeiten sowie die Auslösung des Mass-Loss-Circuit-Breakers.

MUST Bei einem endpunktspezifischen Wegfall und fehlender weiterer gültiger Verfügbarkeitsmeldung MUSS `availabilityMonitoringStatus.state` zunächst auf `unconfirmed` gesetzt werden. Erst nach Ablauf der festgelegten Timeout- und Karenzfristen DARF der `operationalStatus` automatisch auf `down` gesetzt werden, sofern die Überwachung aktiviert, kein manueller Override aktiv und der `lifecycleStatus` nicht `retired` ist.

MUST Die bloße Beendigung oder Verlagerung eines Monitoring-Clients DARF nicht unmittelbar zu einer automatischen Änderung des `operationalStatus` führen. `availabilityMonitoringStatus.state` MUSS zunächst auf `unconfirmed` gesetzt und eine neue vollständige Karenzfrist begonnen werden. Wird innerhalb dieser Frist keine neue gültige Verfügbarkeitsmeldung aufgebaut und arbeitet die Überwachungsinfrastruktur ordnungsgemäß, MUSS der fortdauernde Wegfall anschließend als endpunktspezifischer Wegfall behandelt werden; eine automatische Änderung auf `down` ist dann unter den Voraussetzungen dieses Kapitels zulässig.

MUST Bei einer Störung der Überwachungsinfrastruktur MUSS `availabilityMonitoringStatus.state` auf `monitoringUnavailable` gesetzt werden. Der `operationalStatus` DARF NICHT geändert werden; laufende Timeout- und Karenzfristen MÜSSEN ausgesetzt werden.

MUST Der Verzeichnisdienst MUSS einen clusterweit wirksamen Mass-Loss-Circuit-Breaker implementieren. Solange dieser geöffnet ist, DÜRFEN keine automatischen `down`-Übergänge aufgrund fehlender WebSocket-Verfügbarkeitsmeldungen ausgeführt werden.

MUST Die technische Schnittstellenbeschreibung MUSS Beobachtungsfenster, absolute und relative Auslöseschwelle sowie die Stabilitätszeit vor Rücksetzung des Circuit-Breakers verbindlich festlegen.

MUST Nach Wiederherstellung der Überwachungsinfrastruktur DÜRFEN während der Störung abgelaufene Fristen NICHT rückwirkend angewendet werden. Für Einträge ohne wiederhergestellte Verfügbarkeitsmeldung MUSS eine neue vollständige Karenz- beziehungsweise Wiederanlauffrist beginnen. Läuft diese Frist bei ordnungsgemäß arbeitender Überwachungsinfrastruktur ohne neue gültige Meldung ab, MUSS der fortdauernde Wegfall als endpunktspezifisch bewertet werden.

5.8.3. Wiederaufnahme der Verfügbarkeit

MAY Wird nach einem Wegfall erneut eine technische Verfügbarkeitsmeldung aufgebaut, KANN der Verzeichnisdienst diese berücksichtigen.

MUST Eine bestätigte neue Verfügbarkeitsmeldung MUSS `availabilityMonitoringStatus.state` auf `confirmed` setzen und laufende Timeout-, Karenz- und Wiederanlauffristen für den betroffenen Eintrag beenden.

MUST Ein zuvor durch diesen Mechanismus automatisch auf `down` gesetzter `operationalStatus` MUSS wieder auf `operational` gesetzt werden, sofern die automatische Statussteuerung aktiviert, kein manueller Override aktiv und der `lifecycleStatus` nicht `retired` ist.

MUST Nach einem Zustand `monitoringUnavailable` DARF die Wiederherstellung der Überwachungsinfrastruktur NICHT rückwirkend zu `down` führen. Maßgeblich sind ausschließlich nach der Wiederherstellung neu eingehende Verfügbarkeitsmeldungen und neu beginnende Fristen.

MUST Ist ein manueller Override aktiv, DARF die technische Verfügbarkeitsüberwachung den `operationalStatus` NICHT verändern; `availabilityMonitoringStatus` MUSS gleichwohl weitergeführt werden.

5.8.4. Nachvollziehbarkeit automatischer Statusänderungen

MUST Automatische Statusänderungen und Änderungen des `availabilityMonitoringStatus` durch den Verzeichnisdienst MÜSSEN nachvollziehbar sein.

SHOULD Hierzu SOLLTEN mindestens folgende Informationen verfügbar sein:

- betroffener Verzeichniseintrag,
- vorheriger und neuer Zustand,
- Zeitpunkt der Änderung,
- Grund und Art der Feststellung,
- verantwortlicher oder auslösender Mechanismus sowie
- bei einem Circuit-Breaker die Auslösewerte, die Anzahl betroffener Einträge und der Zeitpunkt von Öffnung und Rücksetzung.

MUST Ändert sich ausschließlich `availabilityMonitoringStatus.state`, MUSS genau eine neue Revision erzeugt und eine Benachrichtigung mit `eventType = serviceInfoChanged` ausgelöst werden.

MUST Ändern sich `availabilityMonitoringStatus.state` und `operationalStatus` im selben atomaren Vorgang, MUSS genau eine neue Revision erzeugt und eine Benachrichtigung mit `eventType = statusChanged` ausgelöst werden.

MUST Der Empfang eines Heartbeats im bereits bestehenden Zustand `confirmed` DARF weder eine neue Revision noch eine Benachrichtigung auslösen.

6. Quellen

[1] Bundesnetzagentur. Beschluss BK7-24-01-014 in Sachen Wasserstoff-Ausgleichs- und Bilanzierungsgrundmodell (WasABi) vom 27.10.2025. https://www.bundesnetzagentur.de/DE/Beschlusskammern/1_GZ/BK7-GZ/2024/BK7-24-0014/BK7-24-01-0014_Beschluss_Internet.html
[2] Bundesamt für Sicherheit in der Informationstechnik. BSI TR-03116-3 – Kryptographische Vorgaben für Projekte der Bundesregierung, Teil 3: Intelligente Messsysteme, Stand 2025, 13.12.2024. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-3.html
[3] Messstellenbetriebsgesetz (MsbG), § 52 Abs. 4, in der jeweils geltenden Fassung. https://www.gesetze-im-internet.de/messbg/__52.html
[4] BDEW. Regelungen zum Übertragungsweg für API-Webdienste. Version 1.2, Publikationsdatum 01.04.2025, anzuwenden ab 01.10.2025. https://www.bundesnetzagentur.de/DE/Beschlusskammern/BK06/BK6_83_Zug_Mess/835_mitteilungen_datenformate/Mitteilung_51/Mitteilung_Nr_51.html
[5] BDEW. Regelungen zum Verzeichnisdienst. Version 1.1, Publikationsdatum 01.04.2025, anzuwenden ab 01.10.2025. https://www.bundesnetzagentur.de/DE/Beschlusskammern/BK06/BK6_83_Zug_Mess/835_mitteilungen_datenformate/Mitteilung_51/Mitteilung_Nr_51.html
[6] Eastlake, D. Transport Layer Security (TLS) Extensions: Extension Definitions. IETF RFC 6066, January 2011. https://www.rfc-editor.org/info/rfc6066
[7] Thomson, M. Record Size Limit Extension for TLS. IETF RFC 8449, August 2018. https://www.rfc-editor.org/info/rfc8449
[8] Berners-Lee, T., Fielding, R., and Masinter, L. Uniform Resource Identifier (URI): Generic Syntax. IETF RFC 3986, January 2005, insbesondere Kapitel 6.2.2 und 6.2.3. https://www.rfc-editor.org/info/rfc3986
[9] Rundgren, A., Jordan, B., and Erdtman, S. JSON Canonicalization Scheme (JCS). IETF RFC 8785, June 2020. https://www.rfc-editor.org/info/rfc8785
[10] Bundesamt für Sicherheit in der Informationstechnik. Certificate Policy der Smart Metering PKI. Version 1.1.2, 25.01.2023. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03109/PKI_Certificate_Policy.html
[11] Interner Verweis: Teil 1, Kapitel 3.6 sowie Teil 2, Kapitel 4 und 5 dieser Guideline.
[12] GitHub-Organisation H2-AG-API für die Anlage der API-Kennungen: https://github.com/H2-AG-API

Version	Datum	Änderung	Autor
1.00	24.08.2026	Initiale Erstellung	Arbeitsgruppe API