



ECOS TrustManagementAppliance®

ECOS REST API

Dokumentation

Version: 1.1

Datum: 09/2025

IT-Security Solutions

Made in Germany

© by ECOS Technology GmbH

Weitergabe sowie Vervielfältigung dieses Dokuments, Verwendung und Mitteilung seines Inhalts sind verboten, soweit nicht ausdrücklich von uns gestattet. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte für den Fall der Patent- oder Gebrauchsmustereintragungen vorbehalten

Inhalt

1. Einleitung	4
2. Definitionen	5
2.1 API-Versionierung	5
2.2 Uniform Resource Locators	5
2.3 Content-Type	5
2.4 Namenskonventionen	6
2.4.1 Identifier	6
2.4.2 Resource-Type	6
2.4.3 Attribute	6
2.4.4 Standardattribute für alle Objekte	6
2.4.5 Zeit und Datum	8
2.4.6 Internationalisierung	8
2.5 Schemadaten	8
2.6 Authentisierung und Autorisierung	9
2.6.1 Ablauf der Authentisierung	9
2.6.2 Mehr-Faktor-Authentisierung	10
3. CRUD-Schema	11
3.1 CREATE – Erstellen	11
3.1.1 Neue Instanz erstellen	11
3.1.2 Factory-Funktion newinstance	12
3.2 REQUEST – Lesen	13
3.2.1 Abfragen von Resource-Sammlungen	14
3.3 CREATE or UPDATE – Erstellen oder Aktualisieren	18
3.4 UPDATE – Teilaktualisieren	19
3.4.1 Konfliktvermeidung mit _rev	19
3.5 DELETE – Löschen	20
4. Transitionen / Operationen (Funktionsaufrufe)	21
4.1 Schemainformationen für Transitionen	22
5. Relationships	24
5.1 Schemainformationen einer Relationship	26
5.2 Bearbeiten von Relationships	26
5.2.1 Zu-1-Relationships	27
5.2.2 Zu-N-Relationships	28
5.3 Verschieben von Instanzen eines Resource-Typs	30
5.4 Aggregierte Wertobjekte (value objects)	31
6. Fehler und Fehlermeldungen	34
7. Referenzen	36

1. Einleitung

Die vorliegende Dokumentation beschreibt das RESTful Application Programming Interface (REST API) der ECOS **TrustManagementAppliance**[®].

Mit der Trust Management Appliance (TMA) stellt ECOS eine REST API nach aktuellem Stand der Technik zur Verfügung, um sowohl das Frontend anzubinden als auch eine marktgängige Schnittstelle zur M2M-Kommunikation anzubieten.

Die ECOS REST API ermöglicht die Integration der TMA und ihrer Funktionen in eine gegebene Infrastruktur sowie die Verwaltung der damit verbundenen Kontroll- und Produktionsprozesse. Sie erlaubt den Zugriff auf dieselben Funktionen und Informationen, die auch über die Administrationsoberfläche der TMA verfügbar sind.

Diese Dokumentation richtet sich an Entwickler und Systemtechniker, die für die Steuerung und Integration von ECOS Appliances verantwortlich sind.

2. Definitionen

2.1 API-Versionierung

Die Versionsnummer ist Bestandteil des URI-Blocks des API-Endpoint und ist aktuell v2.0.

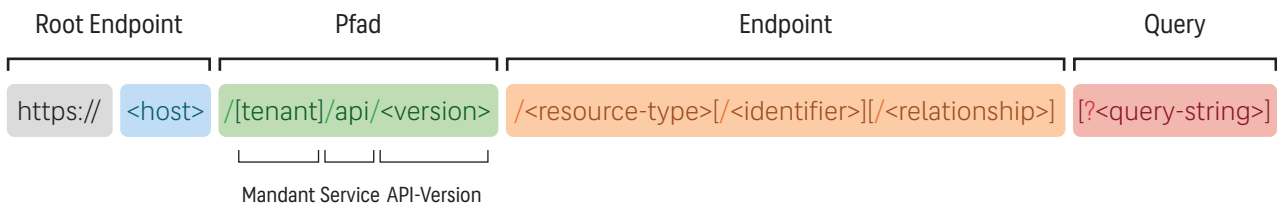
Die API-Versionierung orientiert sich nicht an der Softwareversion bzw. der Versionierung der Softwarekomponenten, sondern repräsentiert den Schnittstellenvertrag der REST API. Abwärtskompatible funktionale Ergänzungen inkrementieren die Nebenversionsnummer, fundamentale Veränderungen werden durch eine neue Hauptversionsnummer ausgedrückt.

Die verfügbaren Versionen werden bis zu ihrer jeweiligen Abkündigung parallel zur Verfügung gestellt. Eine DNS- oder HTTP-Header-basierte Versionierung wird nicht unterstützt.

2.2 Uniform Resource Locators

Die ECOS REST API bildet ein durchgängiges `id+type`-Schema auf dem Resource-Locator-Schema ab.

URLs für REST API-Requests werden gem. folgendem Schema referenziert:



Syntax	
<code><host></code>	DNS-Name des Zielhosts bzw. des API-Gateways
<code>[tenant]</code>	Mandant, für den die REST API bereitgestellt wird Die Angabe ist optional bei Single-Tenant-Installationen
<code>api</code>	Servicepfad Trennt API-Requests von anderen Ressourcen des Mandanten, z.B. statische HTML-Seiten oder Applikationen
<code><version></code>	Angesprochene API-Version: v<Major-Number>.<Minor-Number>
<code><resource-type></code>	Typ der angesprochenen Ressource Dieser kann als von der REST API verwendeter Klassenname betrachtet werden (→ siehe Abschnitt 2.4 Namenskonventionen).
<code>[identifier]</code>	ID einer Instanz der Ressource Verweist auf die eindeutige ID des Dokuments.
<code>[function]</code>	Komponenten einer Instanz z.B. <code>relationships</code> (→ siehe Abschnitt 5. Relationships).
<code>[query-string]</code>	Wird bei GET-Anfragen zur Qualifizierung verwendet, z.B. von Suchkriterien

2.3 Content-Type

Als Content-Type wird `application/vnd.api+json` verwendet. Die ECOS REST API bietet aktuell keinen XML-Support.

2.4 Namenskonventionen

2.4.1 Identifizier

Jede Ressource hat eine `id`, die zusammen mit dem Resource-Typ die Ressource eindeutig bezeichnet.

Beispiel: `"id": "397378a877bbb59a23d0620abd00eed9"`

2.4.2 Resource-Type

`type` beschreibt Ressourcen, die gemeinsame Attribute und Beziehungen verwenden. Die Kombination `type` + `id` ist über alle Ressourcen hinweg immer eindeutig. Ein Resource-Typ enthält nur Kleinbuchstaben. Wörter werden mit einem Minus oder einem Unterstrich gekoppelt.

Beispiel: `"type": "user_account"`

2.4.3 Attribute

Zeichenklasse

```
^[A-Za-z_][A-Za-z0-9_]*
```

Attribute folgen den gängigen Konventionen für Bezeichner, beginnen mit einem Buchstaben oder einem Unterstrich und enthalten weder Klammern, Satzzeichen, Trenner noch sonstige Sonderzeichen. Einige Attribute sind für die Verwendung durch das darunterliegende System reserviert.

Hinweis

Bei der ECOS HTTP API (v1) wurden Attributnamen mit einem Punkt als Präfix verwendet. Diese Präfixpunkte entfallen bei der ECOS REST API (v2) ersatzlos.

2.4.4 Standardattribute für alle Objekte

> `aclset`

Verweist auf die mit einer Ressource verbundenen Berechtigungen.

Beispiel:

```
"aclset": {
  "data": [
    {
      "attributes": {
        "acl": false,
        "admin": false,
        "browse": true,
        "browseparent": true,
        "create": true,
        "delete": false,
        "execute": false,
        "read": true,
        "secrets": false,
        "tochilids": true,
        "write": true
      },
      "type": "user_account-aclset"
    }
  ]
}
```

> **cn**

Der Common Name der Ressource hat beschreibenden Charakter und ist frei wählbar. Er sollte so gesetzt werden, dass die Bedeutung der Ressource aus ihm hervorgeht.

Beispiel: `"cn": "OCSP-Dienst"`

> **creators_name**

Verweist auf den Namen des Benutzers, der die Ressource erstellt hat.

Beispiel: `"creators_name": "Ann Schaulich"`

> **create_timestamp**

Verweist auf Datum und Uhrzeit der Erstellung einer Ressource.

Beispiel: `"create_timestamp": "2025-04-19T12:47:33Z"`

> **docpath**

Verweist auf den Dokumentpfad innerhalb einer Datenbank und besteht aus einer hierarchischen, kommasetrennten Liste aller übergeordneten IDs einer Ressource, beginnend mit der ID des Wurzelements.

Beispiel: `"docpath": "6742e597d1d3742470dc45952d000a34,6742e597d1d3742470dc45952d00264b"`

> **docroot**

Bezeichnet das Wurzelement einer Datenbank. Es wird nur in einer Ressource gesetzt. Wenn **docroot** gesetzt ist, darf kein **docpath** vorhanden sein.

Beispiel: `"docroot": "root"`

Hinweis

Das **docroot**-Attribut steht nicht für alle Resource-Typen zur Verfügung.

> **modify_timestamp**

Verweist auf Datum und Uhrzeit der letzten Änderung der Ressource.

Beispiel: `"modify_timestamp": "2024-03-11T12:47:33Z"`

> **modifiers_name**

Verweist auf den Namen des Benutzers, der die Ressource zuletzt geändert hat.

Beispiel: `"modifiers_name": "Alice"`

> **_rev**

Verweist auf die Revisionsnummer der Ressource. Die Revisionsnummer kann ein beliebiger String sein. Soweit vorhanden, ist sichergestellt, dass sich die Revision bei jeder Änderung der Ressource ebenfalls ändert (→ siehe Abschnitt **3.4.1 Konfliktvermeidung mit _rev**).

Beispiel: `"_rev": "2-69c1474ab819c8aacf42d7c961aa84ed"`

Hinweis

Nicht alle Resource-Typen verfügen über eine Revision.

2.4.5 Zeit und Datum

Format

```
YYYY-MM-DDThh:mm:ssZ
```

Zeit- und Datumsformate folgen dem Standard **RFC 3339** und werden als UTC-Zeitstempel angegeben.

Beispiel: "cert_not_after": "2034-03-18T11:13:31Z"

2.4.6 Internationalisierung

Endpoints

```
https://<host>/api/<version>/<resource-type>/help
```

```
https://<host>/api/<version>/<resource-type>/translations
```

Alle verfügbaren Texte und Übersetzungen können über die Endpoints `/<resource-type>/help` bzw. `/<resource-type>/translations` abgerufen werden.

Beispiel:

```
GET https://tma.domain.de/api/v2.0/role/translations
Response 200 OK
{
  "attr:sfbase_own_template": "",
  "attr:doctag": "Dokument-Tag",
  "val:stash": "Daten sichern",
  "attr:aclgroup": "Sicherheitsgruppe",
  "attr:sfbase_apply_template": "",
  "attr:tochilids": "Vererben",
  "attr:browse": "Browsen",
  "attr:ou_models": "Zugehöriger Vorlagencontainer",
  "attr:ou_host_config": "Zugehöriges System",
  "attr:docroot": "Wurzelobjekt",
  ...
}
```

2.5 Schemadaten

JSON-Schema-Definition der Resource-Typen

```
https://<host>/[tenant]/api/<version>/<resource-type>/schema
```

OpenAPI-Definition der Resource-Typen

```
https://<host>/[tenant]/api/<version>/<resource-type>/openapi
```

Schemadaten werden gemäß JSON-Schema und OpenAPI bereitgestellt. Bei Aufruf des OpenAPI-Endpoint im Browser (Accept: text/html) wird die OpenAPI-Spezifikation als HTML-Dokument gerendert. Für die JSON-Darstellung im Browser kann folgende URL verwendet werden:

```
https://<host>/[tenant]/api/<version>/<resource-type>/openapi?-datatype=jsonpretty
```

Ansonsten hängt das Format, in dem die OpenAPI-Spezifikationen geliefert werden, vom angeforderten Content-Type ab.

i Information

Die OpenAPI-Definitionen sind in der Expertenansicht der TMA im Admin-Reiter eines Konfigurationsobjekts unter **Admin-Info** → **OpenAPI definition** direkt abrufbar. Unter **Admin-Info** → **RestAPI Result** kann zusätzlich der Inhalt der Ressource, so wie er von der REST API zurückgeliefert wird, eingesehen werden. → Siehe Abschnitt **12.6 Werkzeuge des Systems** im Administrator-Handbuch der ECOS TrustManagementAppliance®.

2.6 Authentisierung und Autorisierung

Die ECOS REST API verwendet ein Token-basiertes Authentisierungsverfahren. Der Einstiegspunkt ist der Endpunkt `/logons` der Authentisierungs-API.

Der Logon-Aufruf erfordert keine vorherige Authentisierung. Bei erfolgreichem Logon stellt der Server ein Access Token aus.

Alle weiteren API-Aufrufe müssen dieses Access Token im HTTP-Header mitführen.

2.6.1 Ablauf der Authentisierung

2.6.1.1 Logon anfordern

Der Client übermittelt seine Zugangsdaten (Credentials) per POST an den Endpunkt `/logons`.

Beispiel-Request (JSON-Body):

```
POST https://{server}/api/v2.0/logons
REQUEST-BODY JSON
{
  "data": {
    "attributes": {
      "password": "{password}",
      "user": "{user}"
    },
    "type": "logons"
  }
}
```

Bei erfolgreicher Authentisierung liefert der Server ein JSON-Objekt zurück, das unter anderem das Access Token im Attribut „authorization“ enthält:

```
RESPONSE-STATUS 200
RESPONSE-HEADER [
  "Set-Cookie": "eyJh...",
  ...
]
RESPONSE-BODY JSON
{
  "data": {
    "type": "logons",
    "id": "a59cfc7a67a21a9b3889fc732916974b",
    "attributes": {
      "authorization": "Bearer eyJh...",
      "not_after": "2025-02-26T08:25:55Z",
      "expires_in": 900
    }
  },
  "links": {
    "self": "/logons/1",
    "refresh": {
      "href": "/aas/api/v2.0/logons/a59cfc7a67a21a9b3889fc732916974b/actions",
      "meta": {
```

```

    "refresh": {
      "refresh_token": "eyJh..."
    }
  }
}
}
}

```

Zusätzlich wird das Access Token aus Kompatibilitätsgründen auch als Session Cookie im Response-Header bereitgestellt.

Das Access Token muss bei allen weiteren Anfragen im HTTP-Header **Authorization** übermittelt werden:

```

POST https://{{server}}/api/v2.0/cert_ca
REQUEST-HEADER [
  "Authorization": "Bearer eyJh...",
  ...
]
REQUEST-BODY JSON
...

```

2.6.1.2 Token-Erneuerung (Refresh)

Um ein neues Access Token anzufordern, ohne die Zugangsdaten erneut senden zu müssen, kann das Refresh Token verwendet werden. Dazu wird die unter [Links](#) in [refresh.href](#) enthaltene URL per POST aufgerufen. Der Request-Body entspricht dem im `meta`-Attribut zurückgegebenen JSON-Objekt.

2.6.2 Mehr-Faktor-Authentisierung

Die Authentisierungs-API unterstützt u.a. Client-Zertifikate. Dabei wird das Zertifikat bereits im TLS-Handshake geprüft.

Um diese Methode zu verwenden, wird beim Logon-Aufruf zusätzlich das Feld `"tls_inherited": true` gesetzt:

```

POST https://{{server}}/api/v2.0/logons
REQUEST-BODY JSON
{
  "data": {
    "attributes": {
      "password": "{{password}}",
      "user": "{{user}}",
      "tls_inherited": true
    },
    "type": "logons"
  }
}

```

Voraussetzungen:

- ◆ Das Benutzerkonto muss mit dem entsprechenden Zertifikat konfiguriert sein.
- ◆ Das Zertifikat muss dem Client zur Verfügung stehen (z. B. auf einer Smartcard).

3. CRUD-Schema

Das Akronym CRUD bezeichnet die vier grundlegenden Operationen, die eine Anwendung ausführen soll: Create, Read, Update, Delete.

Dieses Schema wird mit den Methoden POST, GET, PUT/PATCH und DELETE wie folgt auf HTTP übertragen:

Operation			Methode	HTTP-Status-Codes
C	Create	Erstellen	POST	201
R	Read	Lesen	GET	200
U	Update	Aktualisieren	PUT/PATCH	200 - 201 - 202
D	Delete	Löschen	DELETE	200 - 202 - 204

3.1 CREATE – Erstellen

Syntax

POST `https://<host>/api/<version>/<resource-type>`

* mandatory

Request

Body

```

data *
├── actions
├── attributes
├── id
├── relationships
└── type *
```

Response

Body

```

data *
├── attributes
├── id
├── relationships
└── type *
```

200 201

Mit der POST-Methode wird eine neue Instanz eines Resource-Typs erstellt. Das anzulegende Objekt wird im Body mitgeliefert. Die URL darf keinen Identifier enthalten.

3.1.1 Neue Instanz erstellen

Beispiel – Ein einfacher Request erstellt eine neue Instanz vom Resource-Typ `cert_server` (Zertifikat):

```

POST https://tma.domain.de/api/v2.0/cert_server

{
  "data": {
    "type": "cert_server",
    "attributes": {
      "cn": "Certificate",
      "cert_days": 3650
    }
  }
}
```

In diesem Beispiel wird die neue Instanz zunächst nur als neues Zertifikatdokument mit den angegebenen Werten sowie Standardwerten angelegt, das Zertifikat ist jedoch noch nicht ausgestellt (→ siehe Abschnitt **4. Transitionen / Operationen**).

3.1.2 Factory-Funktion `newinstance`

Query-Parameter

<code>fields</code>	Kommaseparierte Liste der Attribute, die das Resultat enthalten soll.
<code>minfields</code>	Kommaseparierte Liste der Attribute, die das Resultat mindestens enthalten soll. Ist ein Attribut in der Liste, wird es auch dann zurückgeliefert, wenn es keinen Wert hat.

Um das Erstellen neuer Instanzen zu vereinfachen, kann für jeden Resource-Typ die Factory-Funktion `newinstance` mit einem GET-Request aufgerufen werden. Die ECOS REST API liefert eine mit Standardwerten belegte Instanz, die nur noch angepasst und per POST-Request der REST API übergeben werden muss.

Eine über `newinstance` abgerufene Instanz wird im Hintergrund nicht persistiert. Entsprechend erhält die Instanz erst bei Anlage in der Datenbank einen Identifier.

Beispiel:

```

GET https://tma.domain.de/api/v2.0/cert_server/newinstance
Response 200 OK
{
  "data": {
    "attributes": {
      "cert_aia_no_crl": false,
      "cert_aia_no_issuer": false,
      "cert_aia_no_ocsp": false,
      "cert_days": 365,
      "cert_display_status": "",
      "cert_dns_add_ip": false,
      "cert_ec_name": "prime256v1",
      "cert_extusage_critical": false,
      "cert_key_algorithm": "rsa",
      "cert_key_size": 3072,
      "cert_pkcs7_no_issuer": false,
      "cert_renew_margin": "30",
      "cert_sig_hash": "sha256",
      "cert_sign_self": "-9999",
      "cert_usage_critical": false,
      "cert_use_smartcard": "0"
    },
    "id": null,
    "relationships": {
      "children": {
        "links": {
          "related": "/api/v2.0/cert_server/children"
        }
      }
    }
  },
  "type": "cert_server"
}

```

Mit dem Query-Parameter `minfields` kann der Server angewiesen werden, auch Attribute zu berücksichtigen, die nicht zwingend vorgeschrieben sind.

Auf dem gleichen Weg können `relationships` benannt werden, die in einer neuen Instanz zwar noch leer sind, für die aber der `candidates`-Link bezogen werden soll, um die Beziehung direkt beim Erstellen zuweisen und dem Client hierfür eine geeignete Auswahl anbieten zu können (→ siehe Abschnitt **5. Relationships**).

Beispiel:

```
GET https://tma.domain.de/api/v2.0/cert_server/newinstance?minfields=cert_ca
Response 200 OK
{
  "data" : {
    "attributes" : {
      "cert_aia_no_crl" : false,
      "cert_aia_no_issuer" : false,
      "cert_aia_no_ocsp" : false,
      "cert_days" : 365,
      "cert_display_status" : "",
      "cert_dns_add_ip" : false,
      "cert_ec_name" : "prime256v1",
      "cert_extusage_critical" : false,
      "cert_key_algorithm" : "rsa",
      "cert_key_size" : 3072,
      "cert_pkcs7_no_issuer" : false,
      "cert_renew_margin" : "30",
      "cert_sig_hash" : "sha256",
      "cert_sign_self" : "-9999",
      "cert_usage_critical" : false,
      "cert_use_smartcard" : "0"
    },
    "id" : null,
    "relationships" : {
      "cert_ca" : {
        "data" : null,
        "links" : {
          "candidates" : "/api/v2.0/cert_server/newinstance/candidates/_cert_ca"
        }
      }
    },
    "children" : {
      "links" : {
        "related" : "/api/v2.0/cert_server/children"
      }
    }
  },
  "type" : "cert_server"
}
```

3.2 REQUEST – Lesen

Syntax

* mandatory

GET https://<host>/api/<version>/<resource-type>/{{id}}

Request

Body X

Response

200

Body

- ∨ data *
 - attributes
 - id
 - relationships
 - type *

Mit der GET-Methode wird eine Instanz eines Resource-Typs gelesen.

Beispiel – Aufruf eines Benutzerkontos anhand von Resource-Typ und Identifier:

```
GET https://tma.domain.de/api/v2.0/user_account/8dd336e3f64d66e246749a72ac0d861a
Response 200 OK
{
  "data": {
    "attributes": {
      "cn": "Geordi La Forge",
      "creators_name": "Starfleet Administration",
      "description": "Crew Member of U.S.S. Enterprise NCC-1701-D",
      "metadata": {
        "Position": "Head of Engineering"
      },
      "ou": "Engineering",
      "title": "Commander",
      "uid": "geordi.laforge",
      ...
    },
    "id": "8dd336e3f64d66e246749a72ac0d861a",
    "relationships": {...},
    "type": "user_account"
  },
  "links": {...}
}
```

3.2.1 Abfragen von Resource-Sammlungen

Mit GET-Requests können neben einzelnen Instanzen auch mehrere Instanzen eines Resource-Typs als Resource-Sammlung (Collection) abgerufen werden. Die ECOS REST API antwortet mit einem Array von Ressourcenobjekten als Ergebnismenge.

Mithilfe verschiedener Optionen können Ergebnismengen begrenzt werden, um die Antwort gezielt auf die Informationen zu begrenzen, die auch wirklich benötigt werden. Diese Optionen für Ergebnismengen können in einfacher Form als Query-Parameter gesendet werden.

3.2.1.1 Projektion bei Ergebnismengen

Syntax

* mandatory

GET https://<host>/api/<version>/<resource-type>

Request

Body

✗

Response

Collection of objects

Body

```

  data
  | attributes
  | id
  | relationships
  | type

```

200

*

*

Parameter

fields → <attributeName>[<furtherAttributes>]

furtherAttributes → LEER | ,<attributeName>[<furtherAttributes>]

Mit einer Projektionsabfrage kann anstelle der gesamten Ressource eine Teilmenge der Attribute abgerufen werden.

Beispiel:

```

GET https://tma.domain.de/api/v2.0/user_account?fields=title,cn
Response 200 OK
{
  "data": [
    {
      "attributes": {
        "cn": "Data",
        "title": "Commander"
      },
      "id": "6858e17332b0ccc4fc67b1f8c80156aa",
      "type": "user_account"
    },
    {
      "attributes": {
        "cn": "Geordi La Forge",
        "title": "Commander"
      },
      "id": "8dd336e3f64d66e246749a72ac0a73a0",
      "type": "user_account"
    },
    ...
  ]
}

```

3.2.1.2 Paginieren

Syntax

* mandatory

GET https://<host>/api/<version>/<resource-type>

Request

Response

200

Body

✗

Collection of objects

Body

- data
 - attributes
 - id
 - relationships
 - type

*

*

Parameter

- limit → <max Elements>
- offset → <n Elements> (optional, default=0)

Links

- prev vorherige Seite
- next nächste Seite
- first erste Seite
- last letzte Seite

Wenn ein Endpoint eine große Ergebnismenge zurückliefert, kann diese paginiert werden. Der Parameter **limit** gibt hierbei die maximale Anzahl an Elementen pro Seite an. Der Parameter **offset** bestimmt, wo im Datensatz die aktuelle Seite beginnt. Die Antwort liefert auch die Operationen zum Blättern.

links enthalten ggf. weitere Query-Parameter für den beschleunigten, zustandslosen Zugriff.

Beispiel:

```

GET https://tma.domain.de/api/v2.0/user_account?offset=3&limit=3
Response 200 OK
{
  "data": [
    {
      "attributes": {
        "cn": "Jean-Luc Picard",
        "title": "Captain"
      },
      "id": "5d3266ffef24419e6224eb50fb1fc",
      "type": "user_account"
    },
    {..},
    {..}
  ],
  "links": {
    "first": "/api/v2.0/user_account?limit=2&offset=0",
    "last": "/api/v2.0/user_account?limit=2&offset=-3",
    "next": "/api/v2.0/user_account?limit=2&offset=1&cursor=a6bb85d29c268e7214c64d2c0a01aeac",
    "prev": "/api/v2.0/user_account?limit=2&offset=-2&cursor=5d3266ffef24419e6224eb50fe4fc"
  }
}

```

3.2.1.3 Filtern

Syntax

* mandatory

GET https://<host>/api/<version>/<resource-type>

Request

Body

×

Response

200

Collection of objects

Body

∨ data

- attributes
- id
- relationships
- type

*

*

Query-Parameter

query → <queryRestriction>[<furtherRestrictions>]

queryRestriction → <attributeName>[<operator>]<value>

furtherRestriction → LEER | \[<logicalOperator>]<queryRestriction>[<furtherRestrictions>]

Logische Operatoren

and	gleich
or	nicht gleich

Operatoren

eq	=	gleich	neq	!=	nicht gleich
gt	>	höher als	lt	<	kleiner als
ge	>=	höher oder gleich	le	<=	kleiner oder gleich
mt	matches	enthält Regex	nmt	not matches	enthält nicht Regex
wc	wildcard	enthält Wildcard	nwc	not wildcard	enthält nicht Wildcard

Mit Query-Parametern und Operatoren können die von einem Endpoint zurückgegebenen Ergebnismengen sinnvoll gefiltert werden.

Beispiel:

```
GET https://tma.domain.de/api/v2.0/user_account?query=ou[eq]Bridge[and]title[eq]Commander
Response 200 OK
{
  "data": [
    {
      "attributes": {
        "cn": "Data",
        "ou": "Bridge",
        "title": "Commander",
        ...
      },
      "id": "6858e17332b0ccc4fc67b1f8c80156aa",
      "links": {...},
      "relationships": {...},
      "type": "user_account"
    }
  ]
}
```

3.2.1.4 Eindeutige Suche

Syntax

* mandatory

```
GET https://<host>/api/<version>/<resource-type>
```

Request

Response

200

Body

×

Collection of objects

Body

- data
 - attributes
 - id
 - relationships
 - type

*

*

Parameter

```
search → <queryRestriction>[furtherRestrictions]
```

Suchparameter ermöglichen eine eindeutige Suche, die immer genau ein Ergebnisobjekt zurückgibt. Ergibt die Suche keinen Treffer, wird der HTTP-Status-Code 404 Not Found zurückgegeben. Ergibt die Suche mehr als einen Treffer, wird der HTTP-Status-Code 409 Conflict zurückgegeben (→ siehe Abschnitt **6. Fehler und Fehlermeldungen**).

Beispiel:

```
GET https://tma.domain.de/api/v2.0/user_account?search=ou[eq]Engineering&fields=cert_subject,cn,title
Response 200 OK
{
  "data": {
    "attributes": {
      "cn": "Geordi La Forge",
      "title": "Commander"
    },
    "id": "8dd336e3f64d66e246749a72ac0d861a",
    "type": "user_account"
  }
}
```

3.2.1.5 Sortierung

Syntax

* mandatory

GET https://<host>/api/<version>/<resource-type>

Request

Body

✗

Response

Collection of objects

200

Body

∨ data
 attributes
 id
 relationships
 type

*

*

Parameter

| sort → <attributeName>[furtherRestrictions]

Mit dem `sort`-Parameter kann die von einem Endpoint zurückgegebene Ergebnismenge in einer bestimmten Reihenfolge angeordnet werden. Die absteigende Sortierung erfolgt durch ein vorangestelltes Minuszeichen.

Beispiel:

```
GET https://tma.domain.de/api/v2.0/user_account?sort=cn
```

Response 200 OK

```
{
  "data": [
    {user a},
    {user b},
    {user c},
  ]
}
```

3.3 CREATE or UPDATE – Erstellen oder Aktualisieren

Syntax

* mandatory

PUT https://<host>/api/<version>/<resource-type>/{{id}}

Request

Body

∨ data
 actions
 attributes
 id
 relationships
 type

*

*

Response

Body

∨ data
 attributes
 id
 relationships
 type

200

*

*

201

*

*

Mit der PUT-Methode wird eine Instanz eines Resource-Typs vollständig aktualisiert bzw. ersetzt. Die URL muss den Identifier enthalten. Das zu aktualisierende Objekt muss vollständig im Body geliefert werden.

3.4 UPDATE – Teilaktualisieren

Syntax

* mandatory

PATCH `https://<host>/api/<version>/<resource-type>/{{id}}`

Request

Body

```

    data *
      actions
      attributes
      id
      relationships
      type *
```

Response

200

201

Body

```

    data *
      attributes
      id
      relationships
      type *
```

Mit der PATCH-Methode werden lediglich die im Body gelieferten Attribute einer Instanz aktualisiert.

Beispiel – Deaktivieren eines Benutzerkontos durch Aktualisierung des Attributs `user_account_locked`:

```

PATCH https://tma.domain.de/api/v2.0/user_account/5d3266ffef24419e6224ebeb50dd980

{
  "data": {
    "type": "user_account",
    "attributes": {
      "user_account_locked": true
    }
  }
}

Response 200 OK

{
  "data": {
    "type": "user_account",
    "relationships": {...}
    "id": "5d3266ffef24419e6224ebeb50dd980",
    "attributes": {
      "cn": "Ro Laren",
      "user_account_locked": true,
      ...
    }
  },
  ...
}
```

3.4.1 Konfliktvermeidung mit `_rev`

Mit dem `_rev`-Attribut wird die Revisionsnummer einer Instanz referenziert. Das Attribut wird von der ECOS REST API mitgeliefert und muss vom Client mit jeder Update-Operation unverändert wieder mitgeliefert werden. Damit stellt der Server sicher, dass zwischenzeitlich keine Änderung stattgefunden hat.

Hat sich die Instanz zwischenzeitlich geändert, liefert der Server einen HTTP-Status-Code 409 CONFLICT (→ siehe Abschnitt **6. Fehler und Fehlermeldungen**). In diesem Fall muss das Objekt erneut gelesen werden und die Änderung mit den aktuellen Wert des `_rev` Attributs durchgeführt werden.

Wird das Attribut nicht mitgeliefert, wird das Update ohne Berücksichtigung der möglicherweise zwischenzeitlich stattgefundenen Änderungen durch andere Benutzer ausgeführt.

3.5 DELETE – Löschen

Syntax

DELETE `https://<host>/api/<version>/<resource-type>/{{id}}`

Request

Body

×

Response

200

Body

```
meta
id
ok
```

Mit der DELETE-Methode wird eine Instanz eines Resource-Typs gelöscht. Die URL muss den Identifier der zu löschenden Instanz enthalten.

⚠ Hinweis

Ressourcen werden immer kaskadisch gelöscht. Die Operation wird nicht ausgeführt und liefert einen Fehler, wenn dadurch Abhängigkeiten nicht mehr aufgelöst werden können, die von außerhalb der zu löschenden Struktur kommen.

Beispiel:

DELETE `https://tma.domain.de/api/v2.0/cert_server/41a8c9289d2f8408fabbb23fb8000643`

Response 200 OK

```
{
  "meta": {
    "docs": [
      [
        {
          "id": "41a8c9289d2f8408fabbb23fb8000643",
          "rev": "8-e1ca453d912c452b8a454152d0f18ba6",
          "ok": true
        }
      ]
    ],
    "id": "41a8c9289d2f8408fabbb23fb8000643",
    "ok": true
  }
}
```

4. Transitionen / Operationen (Funktionsaufrufe)

Funktionsaufrufe werden als zu erstellende Aktionsressource betrachtet und per POST an die `actions`-Komponente einer Instanz übergeben. Mögliche und erlaubte Operationen werden im Ergebnisobjekt mit dem `links`-Attribut gemäß HATEOAS-Schema mitgeliefert.

Beispiel:

```
GET https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0adf6a?fields=links

Response 200 OK

{
  "data": {
    "attributes": {...},
    "id": "8dd336e3f64d66e246749a72ac0adf6a",
    "relationships": {...},
    "type": "cert_server"
  },
  "links": {
    "create_cert": {
      "href": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0adf6a/actions/create_cert",
      "meta": {
        "ca_key_password": "?",
        "priv_key_password": "?"
      },
      "rel": "action"
    },
    ...
  }
}
```

Parameter werden durch die Objektstruktur immer als Named Parameter geliefert. Die Aktion wird direkt im Attribut benannt und die Argumente als Objekt geliefert:

```
POST https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0adf6a/actions

{
  "create_cert": {
    "ca_key_password": "ca-password",
    "priv_key_password": "pk-password"
  }
}

Response 200 OK

{
  "data": {
    "attributes": {
      "cn": "Certificate 2",
      "cert_status": "generated",
      "cert_archive": {...},
      "cert_valid_until": "2026-03-19T09:30:38Z",
      "cert_not_after": "2026-03-19T09:30:38Z",
      ...
    },
    "relationships": {...},
    "type": "cert_server",
    "id": "8dd336e3f64d66e246749a72ac0adf6a"
  },
  "links": {...}
}
```

4.1 Schemainformationen für Transitionen

Schemainformationen zu einer Transition bzw. Operation werden als ergänzender Dialekt (siehe Beispiel [actions](#)) in den Schemadaten geliefert und geben bekannt, welche Transitionen für einen bestimmten Resource-Typ existieren.

Welche davon für eine konkrete Instanz und deren gegebenen Zustand anwendbar sind, wird durch die mit der Instanz gelieferten [links](#) definiert.

Beispiel – [actions](#) im Schema von `cert_server`:

```

GET https://tma.domain.de/api/v2.0/cert_server/schema
Response 200 OK
{
  "description": "Description",
  "properties": {
    "data": {
      "properties": {
        "actions": {
          "properties": {
            "create_cert": {...},
            "create_cert_hold": {...},
            "create_csr": {...},
            ...
            "revoke": {
              "description": "Description",
              "properties": {
                "ca_key_password": {
                  "description": "Description",
                  "title": "CA password",
                  "type": "string"
                }
              }
            },
            "title": "Revoke",
            "type": "object",
            "writeOnly": true
          },
          "sign": {...},
          "stash": {...},
          "unstash": {...},
          "update": {...}
        },
        "title": "cert_server - Certificate: Actions",
        "type": "object",
        "writeOnly": true
      },
      "attributes": {...},
      "id": {...},
      "relationships": {...},
      "type": {...}
    },
    "required": [...]
  }
},
"required": [...],
"title": "cert_server - Certificate",
"type": "object",
...
}

```

Beispiel – `actions` in einer konkreten Instanz von `cert_server`:

```

GET https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac06cc4f
Response 200 OK
{
  "data": {
    "attributes": {...},
    "id": "8dd336e3f64d66e246749a72ac06cc4f",
    "relationships": {...},
    "type": "cert_server"
  },
  "links": {
    "destroy_private_key": {
      "href": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac06cc4f/actions/destroy_private_key",
      "rel": "action"
    },
    "hold": {
      "href": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac06cc4f/actions/hold",
      "meta": {
        "ca_key_password": "?"
      },
      "rel": "action"
    },
    "revoke": {...},
    "self": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac06cc4f",
    "stash": {...},
    "unstash": {...},
    "update": {...}
  }
}

```

5. Relationships

Schema für die Beziehungsentität

```
https://<host>/api/<version>/<resource-type>/<id>/relationships/<relationship-name>
```

Ressourcen sind nicht isoliert, sondern stehen in Beziehung zu anderen Ressourcen. Diese Beziehungen werden durch Relationship-Objekte ausgedrückt. Jedes Element einer Relationship stellt eine Beziehung von dem Ressourcenobjekt, in dem es definiert wurde, zu anderen Ressourcenobjekten dar.

Ein Relationship-Objekt enthält folgende **links**:

- > **candidates** liefert alle Ressourcen, die als Kandidaten für die bezeichnete Beziehung mit der aktuellen Ressource in Frage kommen.
- > **related** liefert das Dokument der referenzierten Ressource, die in Beziehung zur aktuellen Ressource steht.
- > **self** liefert die Beziehung selbst. Mit diesem Link kann eine Relationship direkt verwaltet werden. Das Entfernen einer Beziehung über die Relationship-URL z.B. hebt die Beziehung auf, ohne die Ressource selbst zu löschen.

Jede Ressource enthält außerdem zwei Standard-Relationships:

- > **children** verweist auf Ressourcen, die der aktuellen Ressource untergeordnet sind.
- > **parent** verweist auf übergeordnete Ressourcen, denen die aktuelle Ressource zugeordnet ist.

Beispiele für Relationships-Links

Aufruf der Relationships eines Zertifikats

Die Antwort enthält eine Relationship mit der Beziehungsentität `cert_ca`, der ID `5d3266ffffef24419e6224eb511b59b` und dem Label `Bridge controls`:

```
GET https://tma.domain.de/api/v2.0/cert_server/a6bb85d29c268e7214c64d2c0a018ed7/relationships
Response 200 OK
{
  "cert_ca": {
    "data": {
      "id": "5d3266ffffef24419e6224eb511b59b",
      "label": "Bridge controls",
      "type": "document"
    },
    "links": {
      "candidates": "/api/v2.0/cert_server/a6bb85d29c268e7214c64d2c0a018ed7/candidates/_cert_ca",
      "related": "/api/v2.0/document/5d3266ffffef24419e6224eb511b59b",
      "self": "/api/v2.0/cert_server/a6bb85d29c268e7214c64d2c0a018ed7/relationships/cert_ca"
    }
  },
  "cert_owner": {
    "data": {
      "id": "6742e597d1d3742470dc45952d00ec08",
      "label": "admin",
      "type": "document"
    },
    "links": {...}
  }
}
```

Abfrage des `candidates`-Links

Der `candidates`-Link liefert eine Liste aller CA-Zertifikate, die als Kandidaten für eine Beziehung mit dem Zertifikat in Frage kommen.

Beispiel:

```
GET https://tma.domain.de/api/v2.0/cert_server/a6bb85d29c268e7214c64d2c0a018ed7/candidates/_cert_ca

Response 200 OK

{
  "data": [
    {
      "id": "15bb04e32dc93b46983a28554a0a7eda",
      "label": "Environment controls",
      "type": "document"
    },
    {
      "id": "15bb04e32dc93b46983a28554a0a9f40",
      "label": "Defensive shield",
      "type": "document"
    },
    {
      "id": "15bb04e32dc93b46983a28554a12c3b2",
      "label": "Primary sensor array",
      "type": "document"
    },
    {
      "id": "6858e17332b0ccc4fc67b1f8c8034cf5",
      "label": "Life support",
      "type": "document"
    },
    {
      "id": "5d3266ffef24419e6224ebeb511b59b",
      "label": "Bridge controls",
      "type": "document"
    }
  ]
}
```

Verwalten der Beziehung mit dem `related`-Link

Der `related`-Link liefert das `data`-Objekt des CA-Zertifikats, mit dem das aktuelle Zertifikat signiert wird:

```
GET https://tma.domain.de/api/v2.0/document/5d3266ffef24419e6224ebeb511b59b

Response 200 OK

{
  "data": {
    "attributes": {
      "cn": "Bridge controls",
      ...
    },
    "id": "5d3266ffef24419e6224ebeb511b59b",
    "relationships": {...},
    "type": "cert_ca"
  },
  links: {...}
}
```

Dieses CA-Zertifikat kann auch über das Standardschema https://tma.domain.de/api/v2.0/cert_ca/5d3266ffef24419e6224ebeb511b59b erreicht werden.

5.1 Schemainformationen einer Relationship

Schlüsselwörter im Schema für Beziehungen

Schlüsselwort	Standardwert		Erklärung
type	null	optional	mandatorisch, falls eine Beziehungsentität existiert
targetType		mandatorisch	Typ der referenzierten Resource Der Wert ist ein Array von Typen mit mind. einem Eintrag
minItems	0	optional	minimale Anzahl der Einträge
maxItems	∞	optional	maximale Anzahl der Einträge
isOrdered	false	optional	gibt bei zu-N-Beziehungen an, ob Einträge bei wiederholten Abfragen zuverlässig in der gleichen Abfolge geliefert werden
isUnique	false	optional	gibt bei zu-N-Beziehungen an, ob gleiche Einträge mehrfach enthalten sein können
isChangeable	true	optional	Einträge können einzeln oder gesamthaft verändert werden
isAddOnly	false	optional	limitiert bei zu-N-Beziehungen verfügbare Operationen auf das Hinzufügen von Einträgen

Schemainformationen zu einer Beziehung bzw. Beziehungsentitäten werden als ergänzender Dialekt in den Schemadaten geliefert.

Beispiel:

```

GET https://tma.domain.de/api/v2.0/cert_ca/schema
Response 200 OK
{
  "description": "Description",
  "properties": {
    "data": {
      "properties": {
        ...
        "relationships": {
          "properties": {
            "sfbase_apply_template": {
              "description": "Description",
              "properties": {...},
              "title": "sfbase_apply_template",
              "x-isOrdered": true,
              "x-targetTypes": [
                "template"
              ]
            }
          }
        },
        ...
      }
    }
  }
}

```

5.2 Bearbeiten von Relationships

Relationships können mithilfe der Methoden PATCH und PUT für die gesamte Ressource mitgeändert werden. Sie können aber auch unabhängig von der Gesamtressource, im Folgenden beschrieben, geändert werden.

5.2.1 Zu-1-Relationships

Zu-1-Relationships werden mit der PATCH-Methode aktualisiert.

Zuweisen einer Beziehung

```
PATCH https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0af2bc/relationships/cert_ca

{
  "data": [
    {
      "id": "5d3266fffe24419e6224ebeb511b59b",
      "type": "cert_ca"
    }
  ]
}

Response 200 OK

{
  "data": {
    "attributes": {
      "cn": "Navigation Console",
      "cert_subject": "cn=Navigation Console,ou=Bridge,o=Enterprise",
      ...
    },
    "type": "cert_server",
    "id": "8dd336e3f64d66e246749a72ac0af2bc",
    "relationships": {
      "cert_ca": {
        "data": {
          "type": "document",
          "id": "5d3266fffe24419e6224ebeb511b59b",
          "label": "Bridge controls"
        },
        "links": {
          "candidates": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0af2bc/candidates/_cert_ca",
          "self": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0af2bc/relationships/cert_ca",
          "related": "/api/v2.0/document/5d3266fffe24419e6224ebeb511b59b"
        }
      }
    },
    ...
  }
}
```

Löschen einer Beziehung:

```
PATCH https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac0af2bc/relationships/cert_ca

{
  "data": []
}

Response 200 OK

{
  "data": {
    "type": "cert_server",
    "relationships": {
      "parent": {...},
      "children": {...},
      "cert_owner": {...},
    },
    "attributes": {...},
    "id": "8dd336e3f64d66e246749a72ac0af2bc"
  }
}
```

5.2.2 Zu-N-Relationships

Ersetzen aller Einträge

Ein PATCH-Request ersetzt alle Instanzen einer Beziehungsentität:

```
PATCH https://tma.domain.de/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role
{
  "data": [
    {
      "id": "5d3266ffef24419e6224eb512a9b0",
      "type": "role"
    },
    {
      "id": "8dd336e3f64d66e246749a72ac0b7349",
      "type": "role"
    }
  ]
}

Response 200 OK

{
  "links": {...},
  "data": {
    "attributes": {...},
    "id": "6858e17332b0ccc4fc67b1f8c80156aa",
    "type": "user_account",
    "relationships": {
      "role": {
        "links": {
          "candidates": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/candidates/_role",
          "related": [
            "/api/v2.0/document/5d3266ffef24419e6224eb512a9b0",
            "/api/v2.0/document/8dd336e3f64d66e246749a72ac0b7349"
          ],
          "self": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role"
        },
        "data": [
          {
            "label": "Operations Officer",
            "type": "document",
            "id": "5d3266ffef24419e6224eb512a9b0"
          },
          {
            "label": "Second Officer",
            "type": "document",
            "id": "8dd336e3f64d66e246749a72ac0b7349"
          }
        ]
      }
    },
    ...
  }
}
```

Löschen aller Einträge

Der folgende PATCH-Request löscht alle Einträge:

```
PATCH https://tma.domain.de/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role
{
  "data": []
}
```

Response 200 OK

```
{
  "links": {...},
  "data": {
    "attributes": {...},
    "id": "6858e17332b0ccc4fc67b1f8c80156aa",
    "type": "user_account",
    "relationships": {
      "cert_owner": {...},
      "children": {...},
      "parent": {...}
    }
  }
}
```

Hinzufügen einzelner Einträge

Einzelne Einträge werden mit einem POST-Request hinzugefügt:

POST https://tma.domain.de/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role

```
{
  "data": [
    {
      "id": "8dd336e3f64d66e246749a72ac0d417f",
      "type": "role"
    }
  ]
}
```

Response 200 OK

```
{
  "links": {...},
  "data": {
    "attributes": {...},
    "id": "6858e17332b0ccc4fc67b1f8c80156aa",
    "type": "user_account",
    "relationships": {
      "cert_owner": {...},
      "children": {...},
      "parent": {...},
      "role": {
        "links": {
          "candidates": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/candidates/_role",
          "related": [
            "/api/v2.0/document/5d3266ffef24419e6224eb512a9b0",
            "/api/v2.0/document/8dd336e3f64d66e246749a72ac0b7349",
            "/api/v2.0/document/8dd336e3f64d66e246749a72ac0d417f"
          ]
        },
        "self": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role"
      }
    },
    "data": [
      {...},
      {...},
      {
        "label": "Tactical Officer",
        "type": "document",
        "id": "8dd336e3f64d66e246749a72ac0d417f"
      }
    ]
  }
}
```

Löschen einzelner Einträge

Einzelne Einträge werden mit einem DELETE-Request gelöscht, die Instanzen selbst bleiben dabei unberührt:

```
DELETE https://tma.domain.de/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role

{
  "data": [
    {
      "id": "5d3266ffef24419e6224ebeb512a9b0",
      "type": "role"
    }
  ]
}

Response 200 OK

{
  "links": {...},
  "data": {
    "attributes": {...}
    "id": "6858e17332b0ccc4fc67b1f8c80156aa",
    "type": "user_account",
    "relationships": {
      ...
      "role": {
        "links": {
          "candidates": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/candidates/_role",
          "related": [
            "/api/v2.0/document/8dd336e3f64d66e246749a72ac0b7349"
          ],
          "self": "/api/v2.0/user_account/6858e17332b0ccc4fc67b1f8c80156aa/relationships/role"
        },
        "data": [
          {
            "label": "Second Officer",
            "type": "document",
            "id": "8dd336e3f64d66e246749a72ac0b7349"
          }
        ]
      }
    }
  }
}
```

5.3 Verschieben von Instanzen eines Resource-Typs

Ressourcen, für die kein spezifischer Container vorgegeben ist, können durch Anpassen der Relationship `parent` in einen anderen Container verschoben werden.

Beispiel – Verschieben eines Benutzerkontos in einen übergeordneten Container:

```
PATCH https://tma.domain.de/api/v2.0/user_account/8dd336e3f64d66e246749a72ac0a73a0/relationships

{
  "parent": {
    "data": [
      {
        "id": "6742e597d1d3742470dc45952d00264b",
        "type": "document"
      }
    ]
  }
}
```

Response 200 OK

```

{
  "links": {...},
  "data": {
    "attributes": {...},
    "type": "user_account",
    "id": "8dd336e3f64d66e246749a72ac0a73a0",
    "relationships": {
      "parent": {
        "data": {
          "type": "document",
          "id": "5d3266fffe24419e6224ebeb50db5ce"
        },
        "links": {
          "related": "/api/v2.0/document/5d3266fffe24419e6224ebeb50db5ce"
        }
      },
      ...
    }
  }
}

```

5.4 Aggregierte Wertobjekte (value objects)

Value objects oder Wertobjekte speichern einfach oder komplex strukturierte Informationen, die in jedem Fall Bestandteil des besitzenden Objekts sind und keine eigene Identität haben, folglich auch keine Entität darstellen. Wertobjekte werden in der ECOS REST API genutzt, um verschiedene, ggf. variierende Daten in andere Ressourcen einzubetten. Das kommt z.B. bei Policies oder Metadaten zum Einsatz.

Wertobjekte werden im Schema der verwendenden Entität definiert. Die Schemadefinition für Wertobjekte kann aber auch generalisiert und separat abgelegt werden, um darauf aus verwendenden Entitäten zu verweisen und redundante Definitionen zu vermeiden. Wertobjekte können als einzelne Instanz oder als Kollektionen bzw. Array verwendet werden.

Beispiel – Hinzufügen eines Schlüsselwerts für Metadaten:

PATCH https://tma.domain.de/api/v2.0/select_options/8dd336e3f64d66e246749a72ac0acfe1

```

{
  "data": {
    "type": "select_options",
    "attributes": {
      "options": {
        "data": [
          {
            "attributes": {
              "tag": "key",
              "text": "key"
            },
            "type": "select_options-options"
          },
          {
            "attributes": {
              "tag": "newkey",
              "text": "new key"
            },
            "type": "select_options-options"
          }
        ]
      }
    }
  }
}

```

Response 200 OK

```

{
  "data": {
    "relationships": {...},
    "type": "select_options",
    "id": "8dd336e3f64d66e246749a72ac0acfe1",
    "attributes": {
      "options": {
        "data": [
          {
            "type": "select_options-options",
            "attributes": {
              "text": "key",
              "tag": "key"
            }
          },
          {
            "type": "select_options-options",
            "attributes": {
              "tag": "newkey",
              "text": "new key"
            }
          }
        ]
      }
    },
    "links": {
      "newinstance": "/api/v2.0/select_options/5d3266ffef24419e6224ebeb511d5a1/newinstance/options"
    }
  },
  ...
}

```

Da es sich bei Wertobjekten um ein Attribut der verwendenden Instanz handelt, müssen sie als "atomarer Wert" in POST-, PUT- oder PATCH-Requests übergeben werden – unabhängig davon, ob es sich um einen einzelnen Wert oder um eine Kollektion handelt.

Beispiel – Hinzufügen eines neuen Metadatenschlüssels zu einem Zertifikat:

PATCH https://tma.domain.de/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac049f2c

```

{
  "data": {
    "type": "cert_server",
    "attributes": {
      "metadata": {
        "key1": "value 1",
        "key2": "value 2"
      }
    }
  }
}

```

Response 200 OK

```

{
  "links": {...},
  "data": {
    "type": "cert_server",
    "relationships": {...},
    "attributes": {
      "metadata": {
        "key1": "value 1",
        "key2": "value 2"
      }
    },
    ...
  }
}

```

Wertobjektinstanzen enthalten einen `type`. Somit können Schema- bzw. OpenAPI-Informationen wie für Entitäten abgerufen werden, auch wenn es sich nicht um Entitäten handelt:

```

GET https://tma.domain.de/api/v2.0/select_options-options/schema
Response 200 OK
{
  "description": "Description",
  "properties": {
    "data": {
      "properties": {
        "attributes": {
          "properties": {
            "tag": {
              "description": " Description",
              "items": {},
              "title": "Tag",
              "type": "string"
            },
            "text": {
              "description": " Description",
              "items": {},
              "title": "Expression",
              "type": "string"
            }
          },
          "title": "select_options - Select options: Attributes",
          "type": "object"
        },
        "type": {
          "type": "string"
        }
      },
      "required": [
        "type"
      ]
    },
    "required": [
      "data"
    ],
    "title": "Options",
    "type": "object"
  }
}

```

Wertobjekte können komplex strukturiert werden und weitere Wertobjekte enthalten. Ebenso können Wertobjekte auf Instanzen anderer Entitäten verweisen, also `relationships` enthalten. Die damit ausgedrückte Beziehung ist in jedem Fall unilateral, weil die Instanz einer Entität nicht auf ein Wertobjekt einer anderen Instanz verweisen darf.

6. Fehler und Fehlermeldungen

Error Response

Body

▼ errors

```

args
document_uri
err *
title *

```

**mandatory*

Treten in der Verarbeitung einer Operation innerhalb der ECOS REST API oder dahinterliegend Fehler auf, werden diese systematisch mit einem HTTP-Status 4xx oder 5xx zurückgemeldet. Weiterführende Details werden im Body in einem `errors`-Array zurückgegeben:

```

{
  "errors": [
    {
      "args": "[Additional informations for the error]",
      "document_uri": "string",
      "err": "Error identifier",
      "title": "Human readable error message"
    }
  ]
}

```

Beispiele:

> HTTP 403

STATUS 403 - Forbidden - the caller is not authorized

```

{
  "errors": [
    {
      "result": "forbidden"
    }
  ]
}

```

> HTTP 404

STATUS 404 - Not Found

```

{
  "errors": [
    {
      "document_uri": "/api/v2.0/user_account/a6bb85d29c268e7214c64d2c0a018ed7?",
      "mod": "_Type#358",
      "result": "error",
      "reason": "Object 'user_account' does not exist in database",
      "err": "not_found",
      "title": "Document not found",
      "args": [
        "user_account",
        "a6bb85d29c268e7214c64d2c0a018ed7"
      ]
    }
  ]
}

```

> HTTP 409

STATUS 409 - Conflict

```
{
  "errors": [
    {
      "title": "A conflict with another modification occurred during saving. Please reload object and retry.",
      "err": "result_not_unique",
      "document_uri": "/api/v2.0/user_account?fields=cn,title,role&search=title(eq)Commander",
      "result": "error",
      "args": [
        "***"
      ],
      "mod": "DocumentTable#319",
      "reason": "Result not unique"
    }
  ]
}
```

> HTTP 500

STATUS 500 Internal Server Error

```
{
  "errors": [
    {
      "args": [
        "3368"
      ],
      "err": "validity_greater_than_cas",
      "cn": "New CA certificate",
      "document_uri": "/api/v2.0/cert_server/8dd336e3f64d66e246749a72ac06cc4f/actions?",
      "mod": "cert_ca#722",
      "title": "Server Error",
      "reason": "The validity must be less than the validity of the CA certificate ('New CA certificate': 3368 days)",
      "result": "error",
      "id": "8dd336e3f64d66e246749a72ac092a56"
    }
  ]
}
```

7. Referenzen

Allgemein

RFC2616	Hypertext Transfer Protocol -- HTTP/1.1	https://datatracker.ietf.org/doc/html/rfc2616
JSON:API	Latest Version	https://jsonapi.org/format
JSON-Schema		https://json-schema.org

Beziehung zwischen OpenAPI, JSON:API und JSON-Schema

<https://apisyouwonthate.com/blog/json-api-openapi-and-json-schema>

Fehler und Fehlermeldungen

<https://jsonapi.org/format/#error-objects>

Relationships

<https://jsonapi.org/format/#document-resource-object-relationships>

<https://jax.de/blog/software-architecture-design/restful-apis-richtig-gemacht>

<https://json-schema.org/learn/getting-started-step-by-step#nesting-data-structures>

<https://json-schema.org/learn/getting-started-step-by-step#references-outside-the-schema>

