

# Übersicht

## 2.1.1

[Datentypen](#)

## 2.1.2

[Properties](#)

## 2.1.3

[Entitäten](#)

## 2.1.4

[Links](#)

## 2.1.5

[Entity-Pages](#)

## 2.1.6

[Link-Pages](#)

Das Backend kommuniziert über das REST-Verfahren mit den Applikationen. Deshalb gibt es prinzipiell zwei Möglichkeiten, um Daten auszutauschen: XML oder JSON. In RiP haben wir uns für JSON entschieden. Wir nutzen die Library Genson, um ein automatisiertes Mapping von JSON Daten auf unsere Entitäten und Links zu ermöglichen. Deshalb ist es wichtig, dass der App-Entwickler die korrekte JSON-Syntax kennt, um ein korrektes Mapping zu ermöglichen. Im Grunde haben wir versucht, die JSON-Syntax möglichst nahe am Standard zu halten; allerdings gibt es im Standard keine komfortable Möglichkeit im Umgang mit Dates. Deshalb mussten wir mehrere Möglichkeiten testen und bieten nun die für den Anwendungsfall unseres Backends eleganteste Lösung an, die nicht mit zu viel Aufwand für den Entwickler verbunden ist. In den folgenden Abschnitten soll nun zunächst die Syntax für die verschiedenen Datentypen erläutert werden. Anschließend gehen wir auf die Unterschiede zwischen Entitäten und Links ein. Da es bei RiP möglich ist, so genannte Pages anzufordern, die beispielsweise mehrere Entitäten oder mehrere Links beinhalten, wird deren Syntax abschließend noch aufgezeigt und mit Beispielen erläutert.

## Datentypen

Wichtig in RiP ist, dass jede Property immer ein key-value-Paar ist, wobei der key – der Name der Property – immer ein String ist. Der value hingegen kann unterschiedliche Datentypen annehmen. Bei den Datentypen unterscheiden wir zwischen primitiven Java Datentypen und den Objekten, wie zum Beispiel Locations oder Dates. Nachfolgend wird kurz für alle Möglichkeiten ein Beispiel aufgelistet:

- boolean:  
"foo":true
- int / long:  
"foo":21
- float / double:  
"foo":21.21
- char / String:  
"foo":"bar"
- Location:  
"foo":[12.4,23.3]
- Date:  
"foo":{"type":"Date","value":date.toString()}

In der obigen Liste kann man sehen, dass bei den primitiven Datentypen der gewöhnliche JSON-Standard genutzt wird. Eine Location beziehungsweise Ortsinformation besteht immer aus einer X-Koordinate und einer Y-Koordinate – Längengrad und Breitengrad. Da diese beiden oft als double Werte angegeben werden, ist unsere Syntax dementsprechend ein double Array mit zwei Einträgen. Der erste Eintrag repräsentiert die X-Koordinate, der zweite die Y-Koordinate. Beim Date Objekt haben wir, wie vorher angekündigt, keine einheitliche Darstellung im Standard. Jede Art ist immer mit etwas Zusatzaufwand für den Entwickler verbunden und alle Varianten haben Vor- und Nachteile. Wir haben in dem Fall die für uns eleganteste Variante im Hinblick auf Performance und Aufwand für den Entwickler gewählt. Es ist ein verschachteltes JSON-Objekt nötig, wobei im verschachtelten Objekt zwei Einträge nötig sind. Zum einen ein Eintrag mit dem Key „type“, in dem der Type „Date“ eingetragen werden muss. Hier ist es wichtig, auf die korrekte Schreibweise zu achten. Im zweiten Eintrag benötigen wir den Key „value“, welcher dann das Date-Objekt beinhaltet. RiP nutzt dabei das Format **"EEE MMM dd HH:mm:ss zzz yyyy"** mit **Locale.ENGLISH**. Der Entwickler sollte also um sicher zu gehen, dass die korrekte Locale genutzt wird, mit SimpleDateFormat arbeiten. Die Umwandlung von Date in String sollte also folgendermaßen ablaufen:

```
SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy", Locale.ENGLISH);
String date = sdf.format(new Date());
```

## Properties

Bevor wir zu den vollständigen Entitäten und Links kommen, möchten wir zunächst auf die Properties eingehen. Jede Entität und jeder Link hat die Möglichkeit, Properties zu besitzen. Es ist wichtig, zu wissen, wie man ein JSON-Objekt erstellt, welches mehrere Properties beinhaltet, da es an manchen Endpunkten möglich ist, die Properties einer Entität oder eines Links zu bearbeiten. Hierfür sollte man einfach ein JSON-Objekt anlegen und mit key-value-Paaren füllen. Dabei müssen natürlich die zuvor beschriebenen Vorgaben eingehalten werden, das heißt eine Location muss als Array angegeben werden, während ein Date-Property als verschachteltes JSON-Objekt angelegt werden muss. Dieses JSON-Objekt kann dann an den dafür vorgesehenen PUT-Request-Endpunkten einfach übertragen werden. Ein Properties-JSON-Objekt könnte so aussehen:

```
{
  "birthday": {"value": "Sat Feb 22 16:21:46 CET 2014", "type": "Date"},
  "location": [0.15, 0.45],
  "name": "Mueller",
  "firstname": "Stefan"
}
```

Hier gilt anzumerken, dass ein Link nicht zwingend Properties besitzen muss. Es reicht durchaus einen Link auf eine Entität zeigen zu lassen, ohne dass es Properties geben muss.

## Entitäten

Nachdem zuvor die Syntax der Properties geklärt wurde, kommen wir nun zu den Entitäten. Eine Entität beinhaltet grundsätzlich eine ID, einen Typ, einen Selflink, mehrere Properties und mögliche Links. Der Typ ist vom primitiven Datentyp long. Der Selflink kann genutzt werden, um eine Entität schnell über diese URL anzufordern, dies ist nützlich, wenn per Paging mehrere Entitäten angefordert werden. Die Properties werden, wie bereits erwähnt, als JSON-Objekt angegeben. In einer Entität wird dem Attribut properties dieses JSON-Objekt zugewiesen, sodass es zu einem verschachtelten JSON-Objekt wird. Die Links einer Entität sind ein Array von JSON-Objekten. Die Syntax von Links wird im [nachfolgenden Abschnitt](#) behandelt. Wichtig ist, dass bei allen Endpunkten, bei denen eine Entität vom Entwickler mitgeschickt werden muss - beispielsweise zum Abspeichern und Update von Entitäten - diese Syntax zwingend eingehalten werden muss. Allerdings können Links immer nur über ihre eigenen Endpunkte gesetzt werden, da diese zusätzliche Validierungen benötigen. Grundsätzlich müssen also zuerst die Entitäten und danach die Links extra angelegt werden. Da bei GET-Requests allerdings alle Links mit zurück gegeben werden, wird die Syntax hier gleich mit erklärt. Eine Entität sieht in JSON-Form also folgendermaßen aus:

```
{
  "links": [{"object": "URL_TO_LINKED_ENTITY_1",
    "properties": {
      "prop2": true,
      "prop1": 10.5,
      "prop3": [0.34, 0.23]
    },
    "type": "myFriend"
  }],
  "properties": {
    "birthday": {"value": "Sat Feb 22 16:21:46 CET 2014", "type": "Date"},
    "location": [0.15, 0.45],
    "name": "Mueller",
    "firstname": "Stefan"
  },
  "self": "URL_TO_ENTITY_3",
  "type": 10
  "id": 3
}
```

Abschließend gilt zu sagen, dass der Typ einer Entität nicht vom Backend festgelegt beziehungsweise vorgegeben ist. Jeder Entwickler denkt sich seine eigenen Typ-Bezeichnungen aus. So könnte man zum Beispiel Personen als „type“:10 und Autos als „type“:20 angeben.

## Links

Ähnlich zu den Entitäten beinhaltet jeder Link einen Typ, sowie mehrere Properties. Zusätzlich besitzt ein Link ein Attribut namens „object“ welches die URL zu der Entität, auf die der Link zeigt, beinhaltet. Somit lässt sich auch darüber einfach auf die verknüpfte Entität zugreifen. Der Typ ist im Gegensatz zur Entität vom Datentyp String. So könnte man beispielsweise sagen, die Entität mit der Id 1 „hatAuto“ mit der Id 2. Ein Link kann dieselben Datentypen wie eine Entität für seine Properties nutzen, allerdings ist ein Link kein eigenständiges Objekt, sondern kann nur in Verbindung mit zwei Entitäten existieren. Ein Link sieht also zum Beispiel so aus:

```
{
```

```

"object": "URL_TO_LINKED_ENTITY_1",
"properties": {
    "prop2": true,
    "prop1": 10.5,
    "prop3": [0.34, 0.23]
},
"type": "myFriend"
}

```

Hier gilt anzumerken, dass ein Link nicht zwingend Properties besitzen muss. Es reicht durchaus, einen Link auf eine Entität zeigen zu lassen, ohne dass es Properties geben muss.

## Entity-Pages

Das Backend bietet die Möglichkeit, über manche Endpunkte mehr als eine Entität gleichzeitig anzufordern. Dies wird später bei der Beschreibung der Endpunkte immer explizit erwähnt. Ein solches JSON-Objekt, das mehr als eine Entität beinhaltet, wird von uns als Entity-Page bezeichnet. Eine Entity-Page besteht immer aus den Entitäten, den Navigationlinks, über die man die vorherige, jetzige oder nächste Page anfordern kann, sowie der Anzahl der Entitäten innerhalb der Page und der Gesamtzahl Entitäten auf dem Server für den aktuellen Request. Da jede Entität selbst auch einen Selbstlink beinhaltet, kann man einfach auf diese zugreifen, sofern das erwünscht ist. Eine Page sieht folgendermaßen aus:

```

{
  "entities": [
    {"links": [
      {"object": "URL_TO_LINKED_ENTITY_1",
       "properties": {
         "prop2": true,
         "prop1": 10.5,
         "prop3": [0.34, 0.23]
       },
       "type": "myFriend"
     }],
     "properties": {
       "birthday": {"value": "Sat Feb 22 16:21:46 CET 2014", "type": "Date"}
     }
    ,
    {"location": [0.15, 0.45],
     "name": "Mueller",
     "firstname": "Stefan"
    },
    {"self": "URL_TO_ENTITY_3",
     "type": 10,
     "id": 3
    },
    {"links": [],
     "properties": {
       "birthday": {"value": "Sat Feb 22 16:21:46 CET 2014", "type": "Date"}
     }
    ,
     {"location": [0.15, 0.45],
      "name": "Mustermann",
      "firstname": "Max"
    },
    {"self": "URL_TO_ENTITY_1",
     "type": 10,
     "id": 1
    }],
    "navigationLinks": {
      "next": "",
      "self": "URL_TO_ENTITY_PAGE",
      "prev": ""
    },
    "pageSize": 2,
    "totalSize": 2
  }
}

```

Um eine bessere Übersicht zu schaffen, hier noch eine leere Page:

```
{
  "entities": [],
  "navigationLinks": {
    "next": "",
    "self": "URL_TO_ENTITY_PAGE",
    "prev": ""
  }
}
```

```

    "prev":"",
},
"pageSize":0,
"totalSize":0
}

```

Im Attribut `entities` befindet sich ein Array, in welchem die Entitäten als JSON-Objekt verschachtelt vorhanden sind. `NavigationLinks` ist wiederum ein verschachteltes JSON-Objekt, in welchem die drei weiterführenden URLs zu finden sind. Die Attribute `pageSize` und `totalSize` sind vom primitiven Datentyp `int`.

## Link-Pages

Analog zu den Entity-Pages gibt es auch Endpunkte, über die man mehrere Links gleichzeitig anfordern kann. Zum Beispiel, wenn man alle Links eines bestimmten Typs von Entität x haben möchte. Ein JSON-Objekt mit mehreren Links wird analog von uns als Link-Page bezeichnet. Sie beinhaltet ebenfalls die Navigationlinks, über die man die vorherige, jetzige oder nächste Page anfordern kann. Auch die Anzahl der Links innerhalb der Page sowie die Gesamtzahl aller Links auf dem Server, die zu diesem Request passen, sind vorhanden. Natürlich befinden sich in ihr auch die Links mit all ihren Properties. Eine Page sieht folgendermaßen aus:

```
{
  "links": [{"object": "URL_TO_LINKED_ENTITY_1",
    "properties": {},
    "type": "myFriend"
  },
  {"object": "URL_TO_LINKED_ENTITY_5",
    "properties": {},
    "type": "myFriend"
  },
  {"object": "URL_TO_LINKED_ENTITY_4",
    "properties": {},
    "type": "myFriend"
  },
  {"object": "URL_TO_LINKED_ENTITY_3",
    "properties": {},
    "type": "myFriend"
  }],
  "navigationLinks": {
    "next": "",
    "self": "URL_TO_LINK_PAGE",
    "prev": ""
  },
  "pageSize": 4,
  "totalSize": 4
}
```

Auch hier für eine bessere Übersicht noch eine leere Page:

```
{
  "links": [],
  "navigationLinks": {
    "next": "",
    "self": "URL_TO_LINK_PAGE",
    "prev": ""
  },
  "pageSize": 0,
  "totalSize": 0
}
```

Im Attribut befindet sich ein Array, in dem alle Links als JSON-Objekt verschachtelt vorhanden sind. Mit den restlichen Attributen verhält es sich analog zur Entity-Page. Da nun die Voraussetzungen für die Syntax geklärt wurden, soll im nächsten Kapitel die Query-Sprache behandelt werden. Dabei soll ersichtlich werden, welche Funktionalitäten prinzipiell zur Verfügung stehen.