RFC 8984 : JSCalendar: A JSON Representation of Calendar Data

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 juillet 2021

Date de publication du RFC : Juillet 2021

https://www.bortzmeyer.org/8984.html

Beaucoup d'applications gèrent des agendas, avec des réunions à ne pas oublier, des évènements récurrents, des rendez-vous précis. Le format traditionnel d'échange de ces applications est iCalendar, normalisé dans le RFC 5545 [1] (ou, dans sa déclinaison en JSON, jCal, dans le RFC 7265). Ce RFC propose une syntaxe JSON mais, surtout, un modèle de données très différent, **JSCalendar**. Le but est de remplacer iCalendar.

Les principes de base ? Simplicité (au moins dans les cas simples, car les calendriers sont des bêtes compliquées...), uniformité (autant que possible, une seule façon de représenter un évènement), tout en essayant de permettre des conversions depuis iCalendar (RFC 5545 et RFC 7986), donc en ayant un modèle de données compatible. JSCalendar, comme son nom l'indique, utilise JSON, plus exactement le sous-ensemble i-JSON, normalisé dans le RFC 7493.

Bon, mais pourquoi ne pas avoir gardé iCalendar ? Parce qu'il est trop complexe avec plusieurs formats de date, parce que ses règles de récurrence sont ambigües et difficiles à comprendre, parce que sa syntaxe est mal définie. jCal n'a pas ce dernier problème mais il garde tous les autres. On voit même des logiciels utiliser leur propre représentation en JSON des données iCalendar au lieu d'utiliser jCal. Bref, JSCalendar préfère repartir, sinon de zéro, du moins d'assez loin.

Voici un exemple très simple et très minimal de représentation d'un évènement en JSCalendar :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5545.txt

```
{
     "@type": "Event",
     "uid": "a8df6573-0474-496d-8496-033ad45d7fea",
     "updated": "2020-01-02T18:23:04Z",
     "title": "Some event",
     "start": "2020-01-15T13:00:00",
     "timeZone": "America/New_York",
     "duration": "PT1H"
}
```

Comme avec iCalendar, les données JSCalendar peuvent être échangées par courrier ou avec n'importe quel autre protocole de son choix comme JMAP ou WebDAV.

Avant d'attaquer ce format JSCalendar, rappelez-vous qu'il utilise JSON et que la terminologie est celle de JSON. Ainsi, on nomme « objet » ce qui, selon le langage utilisé, serait un dictionnaire ou un tableau associatif. Ensuite, outre les types de base de JSON, JSCalendar a des types supplémentaires (section 1), notamment :

— `Id`, un identificateur, pour que les objets aient un nom simple et unique (c'est une chaîne de caractères JSON). On peut par exemple utiliser un UUID (RFC 9562). Dans l'exemple ci-dessus, le membre `uid` de l'objet est un `Id`.
— `UTCDateTime`, une date-et-heure (un instant dans le temps) au format du RFC 3339 dans une chaîne de caractères JSON, est obligatoirement en UTC. Un tel type est intéressant pour les évènements internationaux comme une vidéoconférence. Dans l'exemple ci-dessus, le membre `updated` est de ce type.
— `LocalDateTime`, une date-et-heure selon le fuseau horaire local. Un tel type est utile pour les évènements locaux, comme un pique-nique. Dans l'exemple ci-dessus, qui concerne une réunion en présentiel, `start` est de ce type. On a besoin des deux types, à la fois parce que les évènements en distanciel et en présentiel n'ont pas les mêmes besoins, et aussi parce que le décalage entre les deux peut varier. Les calendriers, c'est compliqué et le RFC cite l'exemple de la `LocalDateTime` `2020-10-04T02:30:00` qui n'existe pas à Melbourne car le passage à l'heure d'été fait qu'on saute de 2h à 3h, mais qui peut apparaitre dans les calculs (`start` + une durée) ou bien si les règles de l'heure d'été changent.
— `Duration` est une durée, exprimée comme en iCalendar (`PT1H` pour « une heure » dans l'exemple ci-dessus, ce serait `P1Y` pour une année). Gag : une journée (`P1D`) ne fait pas forcément 24 heures, par exemple si on passe à l'heure d'été pendant cette journée.
— `SignedDuration`, une durée qui peut être négative.
— `TimeZoneId`, un identifiant pour un fuseau horaire, pris dans la base de l'IANA `<https://www.iana.org/time-zones>`, par exemple `Europe/Paris`.
— `PatchObject` est un pointeur JSON (ces pointeurs sont normalisés dans le RFC 6901) qui permet de désigner un objet à modifier.
— `Relation` est un objet, et plus un simple type scalaire comme les précédents. Il permet d'indiquer une relation entre deux objets, notamment vers un objet parent ou enfant..
— `Link` est également une relation et donc un objet, mais vers le monde extérieur. Il a des propriétés comme `href` (dont la valeur est, comme vous vous en doutez, un URI) ou `cid` qui identifie le contenu vers lequel on pointe, dans la syntaxe du RFC 2392.

Les types de JSCalendar figurent dans un registre IANA `<https://www.iana.org/assignments/jscalendar/jscalendar.xml#jscalendar-types>`, qu'on peut remplir avec la procédure « Examen par un expert » du RFC 8126.

Bien, maintenant que nous avons tous nos types, construisons des objets. Il y en a de trois types (section 2) :

— `Event` (l'exemple ci-dessus, regardez sa propriété `@type`) est un évènement ponctuel, par exemple une réunion professionnelle ou une manifestation de rue. Il a une date-et-heure de départ, et une durée.

— `Task` est une tâche à accomplir, elle peut avoir une date-et-heure limite et peut avoir une durée estimée.
— `Group` est un groupe d'évènements JSCalendar.

JSCalendar n'a pas de règles de canonicalisation (normalisation) générales, car cela dépend trop de considérations sémantiques que le format ne peut pas connaitre. Par exemple, JSON permet des tableaux, et JSCalendar utilise cette possibilité mais, quand on veut décider si deux tableaux sont équivalents, doit-on tenir compte de l'ordre des éléments (`[1, 2] == [2, 1]`)? Cela dépend de l'application et JSCalendar ne fixe donc pas de règles pour ce cas. (Un cas rigolo et encore pire est celui des valeurs qui sont des URI puisque la canonicalisation des URI dépend du plan - "*scheme*".)

Quelles sont les propriétés typiques des objets JSCalendar (section 4)? On trouve notamment, communs aux trois types d'objets (évènement, tâche et groupe) :

— `@type`, le type d'objet (`Event` dans l'exemple ci-dessus, un des trois types possibles).
— `uid` (`a8df6573-0474-496d-8496-033ad45d7fea` dans l'exemple), l'identificateur de l'objet (il est recommandé que ce soit un des UUID du RFC 9562).
— `prodID`, un identificateur du logiciel utilisé, un peu comme le `User-Agent:` de HTTP. Le RFC suggère d'utiliser un FPI.
— `updated`, une date-et-heure de type `UTCDateTime`.
— `title` et `description`, des chaînes de caractères utiles.
— `locations` (notez le S) qui désigne les lieux physiques de l'évènement. C'est compliqué. Chaque lieu est un objet de type `Location` qui a comme propriétés possibles un type (tiré du registre des types de lieux `<https://www.iana.org/assignments/location-type-registry/location-type-registry.xml#location-type-registry-1>` créé par le RFC 4589), et une localisation sous forme de latitude et longitude (RFC 5870).
— Tous les évènements ne sont pas dans le monde physique, certains se produisent en ligne, d'où le `virtualLocations`, qui indique des informations comme l'URI (via la propriété du même nom). Ainsi, une conférence en ligne organisée par l'assocation Parinux `<https://parinux.org/>` via BigBlueButton aura comme `virtualLocations` `{"@type": "VirtualLocation", "uri": "https://bbb.parinux.org/b/ca--xgc-4r3-n8z", ...}`.
— `color` permet de suggérer au logiciel une couleur à utiliser pour l'affichage, qui est une valeur RGB en hexadécimal ou bien un nom de couleur `<https://www.w3.org/TR/css-color-3/>` CSS.

Et je suis loin d'avoir cité toutes les propriétés possibles, sans compter celles spécifiques à un type d'objet.

Pour le cas de `locations`, le RFC fournit un exemple de vol international (quand il était encore possible de prendre l'avion) :

```
{
  "...": "",
  "title": "Flight XY51 to Tokyo",
  "start": "2020-04-01T09:00:00",
  "timeZone": "Europe/Berlin",
  "duration": "PT10H30M",
  "locations": {
    "418d0b9b-b656-4b3c-909f-5b149ca779c9": {
      "@type": "Location",
      "rel": "start",
      "name": "Frankfurt Airport (FRA)"
    },
    "c2c7ac67-dc13-411e-a7d4-0780fb61fb08": {
      "@type": "Location",
      "rel": "end",
      "name": "Narita International Airport (NRT)",
      "timeZone": "Asia/Tokyo"
    }
  }
}
```

Notez les UUID pour identifier les lieux, et le changement de fuseau horaire entre le départ et l'arrivée. Si l'évènement a lieu à la fois en présentiel et en distanciel (ici, le concert est dans un lieu physique identifié par ses coordonnées géographiques, mais aussi diffusé en ligne), cela peut donner :

```
{
  "...": "",
  "title": "Live from Music Bowl: The Band",
  "description": "Go see the biggest music event ever!",
  "locale": "en",
  "start": "2020-07-04T17:00:00",
  "timeZone": "America/New_York",
  "duration": "PT3H",
  "locations": {
    "c0503d30-8c50-4372-87b5-7657e8e0fedd": {
      "@type": "Location",
      "name": "The Music Bowl",
      "description": "Music Bowl, Central Park, New York",
      "coordinates": "geo:40.7829,-73.9654"
    }
  },
  "virtualLocations": {
    "1": {
      "@type": "VirtualLocation",
      "name": "Free live Stream from Music Bowl",
      "uri": "https://stream.example.com/the_band_2020"
    }
    },
    ...
```

Les fournisseurs de logiciel peuvent ajouter des propriétés définies par eux. Dans ce cas, le RFC recommande fortement qu'ils les nomment en les faisant précéder d'un nom de domaine identifiant le fournisseur. Si celui-ci a `example.com` et veut une propriété `toto`, il la nommera `example.com:toto`. C'est évidemment une solution temporaire, les fournisseurs ont tout intérêt à enregistrer ces propriétés pour qu'elles puissent servir à tout le monde. Le mécanisme d'enregistrement de nouvelles propriétés est « Examen par un expert » (RFC 8126) et les propriétés sont dans un registre IANA <https://www.iana.org/assignments/jscalendar/jscalendar.xml#jscalendar-properties>.

Passons maintenant à un aspect compliqué mais indispensable des calendriers : les règles de récurrence, comme « Réunion de service tous les premiers lundis du mois à 10 h, sauf jour férié ». Il est important de maintenir ces évènements récurrents sous forme de règles, et de ne pas de les instancier immédiatement, car l'application des règles peut donner des résultats différents dans le futur. JSCalendar permet une propriété `recurrenceRules` qui décrit ces règles de récurrence et évidemment une `excludedRecurrenceRules` car s'il n'y a pas d'exceptions, ce n'est pas drôle. Exprimer les règles n'est pas facile. L'idée est de spécifier la récurrence sous forme d'une série de règles, chacune indiquant une fréquence (annuelle, mensuelle, etc), à chaque fois à partir de la propriété `start`, le calendrier utilisé (qui doit être un des calendriers disponibles dans CLDR), la marche à suivre quand une règle produit une date invalide dans ce calendrier (annuler l'évènement, le mettre avant, le mettre après), et plein d'autres propriétés optionnelles comme le jour du mois (« réunion tous les 6 du mois »), de la semaine (« tous les mercredis »), etc. Il faut ensuite plusieurs pages au RFC pour expliquer la façon subtile dont les règles sont appliquées, et se combinent. Les exceptions de la `excludedRecurrenceRules` fonctionnent de la même façon, et sont ensuite soustraites des dates-et-heures sélectionnées par les règles.

Le RFC fournit cet exemple de récurrence : le premier avril arrive tous les ans (et dure une journée, notez le `duration`) :

```
{
 "...": "",
 "title": "April Fool's Day",
 "showWithoutTime": true,
 "start": "1900-04-01T00:00:00",
 "duration": "P1D",
 "recurrenceRules": [{
   "@type": "RecurrenceRule",
   "frequency": "yearly"
 }]
}
```

Alors qu'ici, on fait du yoga une demi-heure chaque jour à 7 h du matin :

```
{
  "...": "",
  "title": "Yoga",
  "start": "2020-01-01T07:00:00",
  "duration": "PT30M",
  "recurrenceRules": [{
    "@type": "RecurrenceRule",
    "frequency": "daily"
  }]
}
```

Bon, ensuite, quelques détails pour aider le logiciel à classer et présenter les évènements. Un évènement peut avoir des propriétés comme `priority` (s'il y a deux réunions en même temps, laquelle choisir ?), `freeBusyStatus` (est-ce que cet évènement fait que je doive être considéré comme occupé ou est-il compatible avec autre chose ?), `privacy` (cet évènement peut-il être affiché publiquement ?), `participationStatus` (je viendrai, je ne viendrai pas, je ne sais pas encore...), et plein d'autres encore.

Il y a aussi des propriétés concernant d'autres sujets, par exemple l'adaptation locale. Ainsi, la propriété `localizations` indique la ou les langues à utiliser (leur valeur est une étiquette de langue du RFC 5646).

Toutes les propriétés vues jusqu'à présent étaient possibles pour tous les types d'objets JSCalendar (évènement, tâche et groupe). D'autres propriétés sont spécifiques à un ou deux types :

— Les évènements (`Event`) peuvent avoir entre autres un `start` (un `LocalDateTime` qui indique le début de l'évènement) et un `duration` (la durée de l'évènement).
— Les tâches (`Task`) peuvent avoir un `start` mais aussi un `due` qui indique la date où elles doivent être terminées, et un `percentComplete` (pour les chefs de projet...).
— Les groupes (`Group`) ont, par exemple, `entries`, qui indique les objets qui sont membres du groupe.

Les fichiers à ce format JSCalendar sont servis sur l'Internet avec le type `application/jscalendar+json`.

Pour terminer, voyons un peu la sécurité de JSCalendar (section 7). Évidemment, toutes les informations stockées dans un calendrier sont sensibles : rares sont les personnes qui accepteraient de voir la totalité de leur agenda être publiée ! Celui-ci permet en effet de connaitre le graphe social (les gens qu'on connait), les lieux où passe la personne, ses habitudes et horaires, bref, que des choses personnelles. Toute application qui manipule des données JSCalendar doit donc soigneusement veiller à la confidentialité de ces données, et doit les protéger.

Le risque d'accès en lecture n'est pas le seul, la modification non autorisée de l'agenda serait également un problème, elle pourrait permettre, par exemple, de faire déplacer une personne en un lieu donné. D'autres conséquences d'une modification pourraient toucher la facturation (location d'une salle pendant une certaine durée) ou d'autres questions de sécurité (activer ou désactiver une alarme à certains moments). L'application qui manie du JSCalendar doit donc également empêcher ces changements non autorisés.

Notez que ces problèmes de sécurité ne concernent pas le format à proprement parler, mais les applications qui utilisent ce format. Rien dans JSCalendar, dans ce RFC, ne le rend particulièrement vulnérable ou au contraire protégé, tout est dans l'application.

Les règles de récurrence sont complexes et, comme tout programme, elles peuvent entrainer des conséquences imprévues, avec consommation de ressources informatiques associées. Un exemple aussi simple que la session de yoga quotidienne citée plus haut pourrait générer une infinité d'évènements, si elle était mise en œuvre par une répétition systématique, jusqu'à la fin des temps. Les programmes doivent donc faire attention lorsqu'ils évaluent les règles de récurrence.

L'un des buts d'un format standard d'évènements est évidemment l'échange de données. Il est donc normal et attendu qu'une application de gestion d'agenda reçoive des objets JSCalendar de l'extérieur, notamment via l'Internet. On voit souvent du iCalendar en pièce jointe d'un courrier, par exemple. Il ne faut pas faire une confiance aveugle à ces données venues d'on ne sait où, et ne pas tout intégrer dans le calendrier du propriétaire. L'authentification du courrier (par exemple avec DKIM, RFC 6376) aide un peu, mais n'est pas suffisante.

Les fuseaux horaires sont une source de confusion sans fin. Un utilisateur qui n'est pas attentif, lorsqu'on lui dit qu'un évènement a lieu à 10h30 (UTC-5) peut croire que c'est 10h30 de son fuseau horaire à lui. (Et encore, ici, j'ai indiqué le fuseau horaire de manière lisible, contrairement à ce que font la plupart des Étatsuniens, qui utilisent leurs sigles à eux comme PST, ou ce que font les Français qui n'indiquent pas le fuseau horaire, persuadés qu'ils sont que le monde entier est à l'heure de Paris.) Cette confusion peut être exploitée par des méchants qui utiliseraient les fuseaux horaires de manière délibérement peu claire pour tromper quelqu'un sur l'heure d'un évènement. (Dans la série Mad Men, les hommes qui ne supportent pas l'arrivée d'une femme dans le groupe lui donnent des informations trompeuses sur les heures de réunion, pour qu'elle manque ces évènements.)

Où trouve-t-on du JSCalendar à l'heure actuelle ? Mobilizon ne l'a pas encore (l'action « Ajouter à mon agenda » exporte du iCalendar). Fastmail annonce qu'ils gèrent JSCalendar (mais apparemment seulement dans les échanges JMAP, pas avec l'import/export normal). Cyrus l'a aussi (si vous avez des détails, ça m'intéresse). Pour le cas des récurrences, vous avez une intéressante mise en œuvre `<https://pypi.org/project/wadu/>` en Python. Attention, il y a aussi plein de logiciels qui s'appellent « jscalendar » (notamment des "*widgets*" JavaScript pour afficher un calendrier dans un formulaire Web) mais qui n'ont aucun rapport avec ce RFC.