JSON

> [Información] Las palabras clave "**DEBE**", "**NO DEBE**", "**REQUERIDO**", "**DEBERÁ**", "**NO DEBERÁ**", "**DEBERÍA**", "**NO DEBE**", "**recomendada**", "**DEBERÍA**", y "**OPCIONAL**" en este documento se DEBE interpretarse como se describe en el RFC 2119 (https://www.ietf.org/rfc/rfc2119.txt).

Conceptos

Una tabla consiste en un conjunto de filas. Cada fila tiene un conjunto de campos (columnas). Por lo general, esperar que cada fila tiene el mismo conjunto de campos y por lo tanto podemos hablar de los campos de la tabla en su conjunto.

En los casos de las tablas en hojas de cálculo o archivos CSV a menudo interpretamos la primera fila como una fila de cabecera que da los nombres de los campos. Por el contrario, en otras situaciones, por ejemplo, tablas de bases de datos SQL los campos (columnas) se designan de forma explícita.

Para ilustrar esta es una tabla de hoja de cálculo clásico:

```
  A |     B |    C |     D         <--- Fila
 ------------------------------------
Vala | Valb | ValC | Vald         <--- Fila
 ...
```

En *JSON* una mesa sería la siguiente:

```
[
 { "A": valor "B": valor, ...},
 { "A": valor "B": valor, ...},
 ...
]
```

Especificación

Un *JSON* tabla de esquema consiste en:

- una lista de descriptores de campo requerido
- opcionalmente, una descripción de la clave primaria
- opcionalmente, una descripción _key como clave foránea.

Un esquema se describe el uso de *JSON*. Esto podría existir como un documento independiente o puede ser integrado dentro de otra estructura *JSON*, por ejemplo, como parte de una descripción del paquete de datos.

Esquema

Un esquema tiene la siguiente estructura:

```
{
        // “campos” es una lista ordenada de descriptores de campo
        // Uno para cada campo(columna) en la tabla
        "campos": [
                //Un campo descriptor
                {
                        "Name": "nombre del campo (por ejemplo, nombre de la columna)",
                        "Title": "Una etiqueta más agradable legible por humanos o título",
                        "Tipo": "Una cadena que especifica el tipo",
                        "Formato": "Una cadena que especifica un formato",
                        "Descripción": "Una descripción para el campo"
                        ...
                },
                ...más descriptores de campo
        ],
        //(Opcional) Especificación de la clave primaria "PrimaryKey": ...
        //(Opcional) especificación de las claves externas "claves externas": ...
}
```

Es decir, una tabla de esquema *JSON* es:

- un hash, que debe contener una clave campos
- campos debe ser una matriz donde cada entrada de la matriz es un descriptor de campo. (Estructura y uso se describen a continuación)
- el hash pueden contener un primaryKey propiedad (estructura y el uso se especifica a continuación)
- el hash pueden contener un ForeignKeys de propiedad (estructura y el uso se especifica a continuación)
- el disperso pueden contener cualquier número de otras propiedades (no definido en esta especificación)

Los descriptores de campo

Un descriptor de campo es un simple hash de *JSON* que describe un solo campo. El descriptor proporciona documentación legible adicional para un campo, así como la información adicional que se puede utilizar para validar el campo o crear una interfaz de usuario para la entrada de datos.

Como mínimo un descriptor de campo contendrá al menos una clave de nombre, pero puede tener claves adicionales como se describe a continuación:

```
{
 "Name": "nombre del campo (por ejemplo, nombre de la columna)",
 "Title": "Una etiqueta más agradable legible por humanos o título para el campo",
 "Tipo": "Una cadena que especifica el tipo",
 "Formato": "Una cadena que especifica un formato",
 "Descripción": "Una descripción para el campo",
 "limitaciones": {
   # A las limitaciones de descriptor
 }
}
```

- un descriptor de campo DEBE ser un Hash

- el descriptor de Hash campo debe contener una propiedad nombre. Esta propiedad debe corresponder al nombre de campo / columna del archivo de datos (si es que tiene un nombre). Como tal, debería ser único (aunque es posible, pero muy mala práctica, para el archivo de datos para tener varias columnas con el mismo nombre). Además, el nombre debe ser considerado entre mayúsculas y minúsculas. En la práctica, mayúsculas y minúsculas de los nombres puede ser limitante en ciertos escenarios, por lo que los consumidores pueden optar por ignorar mayúsculas y minúsculas para los valores de nombre.

- el Hash descriptor de campo puede contener cualquier número de otras propiedades

- Propiedades específicas que pueden ser incluidos en el hash y cuyo significado se define en esta especificación son:
  - *Título*: Una etiqueta legible por humanos o mejor título para el campo
  - *Descripción*: Una descripción de este campo, por ejemplo, "El receptor de los fondos"
  - Tipo: El tipo de campo (cadena, número, etc.) - ver más abajo para obtener más detalles. Si no se proporciona el tipo que el consumidor debe asumir un tipo de "cadena".
  - *Formato*: Una descripción del formato, por ejemplo, *"DD.MM.YYYY"* para una fecha.
  - *Restricciones*: restricciones de un descriptor que puede ser utilizado por los consumidores para validar los valores de campo

Las restricciones sobre el terreno

Un conjunto de restricciones puede estar asociada con un campo. Estas limitaciones pueden ser utilizados para validar los datos en una tabla de esquema *JSON*. Las restricciones pueden ser utilizados por los consumidores para validar, por ejemplo, el contenido de un paquete de datos, o como un medio para validar los datos que se recogen o se actualizan a través de una interfaz de entrada de datos.

Un descriptor de restricciones es un hash *JSON*. Puede contener cualquiera de las siguientes claves.

- required - Un valor booleano que indica si un campo debe tener un valor en cada fila de la tabla. Una cadena vacía se considera que es un valor perdido.

- minLength - Un entero que especifica la longitud mínima de un valor. tipos de campo compatibles son secuencias, como la cadena y la matriz, y las colecciones que contienen elementos, como objeto.

- maxLength - Un entero que especifica la longitud máxima de un valor. tipos de campo compatibles son secuencias, como la cadena y la matriz, y las colecciones que contienen elementos, como objeto.

- unique - Un booleano. Si es verdad, entonces todos los valores de ese campo debe ser único en el archivo de datos en la que se encuentra. Esto define una clave única para una fila aunque una fila podría tener varios de tales llaves.

- pattern - Una expresión regular que se puede utilizar para poner a prueba los valores de campo. Si la expresión regular coincide con el valor es válido. Los valores serán tratados como una cadena de caracteres.

- minimum - especifica un valor mínimo para un campo. Esto es diferente a minLength que comprueba el número de elementos en el valor. Un valor de restricción mínimo comprueba si un valor de campo es mayor que o igual que el valor especificado. La verificación de rango depende del tipo del campo. Por ejemplo. un campo entero puede tener un valor mínimo de 100; un campo de fecha podría tener una fecha mínima. Si se especifica una restricción de valor mínimo, entonces el descriptor de campo debe contener una clave tipo. tipos de campo compatibles son número entero, número, fecha, hora y fecha y hora.

- maximum – igual que el anterior, pero especifica un valor máximo para un campo.

- enum - Matriz de valores, donde cada valor debe cumplir con el tipo y formato del campo. El valor del campo debe coincidir exactamente con un valor en la matriz de enumeración.

Las limitaciones mencionadas anteriormente también pueden definir una lista de tipos de campo compatibles. Implementaciones DEBE informar de un error si se hace un intento para evaluar un valor contra una restricción incompatible.

Un descriptor de restricciones puede contener múltiples restricciones, en cuyo caso el consumidor debe aplicar todas las restricciones al determinar si un valor del campo es válido.

Un archivo de datos, por ejemplo, una entrada en un paquete de datos, se considera válido si todos sus campos son válidos en función de su tipo y limitaciones declaradas.

Tipos de campos y formatos

Un campo `*tipo*` es una cadena que indica el tipo de este campo.

Un campo `*formato*` es una cadena, al ser una palabra clave que indica un formato para el tipo de campo.

Tanto el *tipo* y el *formato* son opcionales: en un descriptor de campo, la ausencia de una propiedad

tipo indica que el campo es del tipo "*cadena*", y la ausencia de una propiedad de formato indica que el tipo de formato del campo es "*default*".

Los tipos se basan en el conjunto de tipo *JSON*-esquema con algunas adiciones y modificaciones menores.

La lista de tipos y formato es el siguiente:

- **String**: el campo contiene cadenas, es decir, secuencias de caracteres.
- **Number**: el campo contiene números de cualquier tipo, incluyendo decimales.
- **Boolean**: el campo contiene datos booleano (verdadero / falso).
- **Objeto**: el campo contiene datos *JSON*.
- **Array**: the field contains data in *JSON* format arrays.
- **datetime**; **date**; **time**: tiempo: el campo contiene valores temporales tales como fechas, horas y fecha-hora.
- **Duración**: una duración de tiempo.
- **Geopunto**: el campo contiene datos que describen un punto geográfico
- **Geojson**: the field contains a *JSON* object according to GeoJSON or TopoJSON spec
- **Any**: Se acepta cualquier *tipo* o *formato*

Clave principal (*primaryKey*)

Una clave principal es un campo o conjunto de campos que identifica de forma única cada fila de la tabla.

La entrada *primaryKey* en el Hash esquema es opcional. Si está presente se especifica la clave principal de esta tabla.

El primaryKey, si está presente, debe ser:

- O bien: una matriz de cadenas con cada secuencia correspondiente a uno de los valores de nombre de campo en la matriz de campos (que denotan que la clave principal se compone de aquellos campos). Es aceptable tener una matriz con un solo valor (lo que indica un solo campo en la clave principal). En sentido estricto, el orden de los valores de la matriz no importa. Sin embargo, se recomienda que se siga el orden de los campos en los campos tiene como aplicaciones cliente pueden utitlize el orden de la lista de clave principal (por ejemplo, en la concatenación de los valores juntos).

- O bien: una única cadena que corresponde a uno de los valores de nombre de campo en la matriz de campos (lo que indica que este campo es la clave principal). Tenga en cuenta que esta versión corresponde a la forma de matriz con un único valor (y puede ser visto como simplemente una forma más conveniente de especificar una única clave principal de campo).

He aquí un ejemplo:

```
"campos": [
 {
  "nombra un"
 },
```

```
 ...
],
"PrimaryKey": "a"
```

He aquí un ejemplo con una clave de la matriz primaria:

```
"Esquema": {
"campos": [
 {
  "nombra un"
 },
 {
  "Name": "b"
 },
 {
  "Name": "c"
 },
 ...
],
"PrimaryKey": [ "a", "c"]
}
```

Llaves externas (*foreignKey*)

Las claves externas, por necesidad, deben ser capaces de hacer referencia a otros objetos de datos. Estos objetos de datos requieren una estructura específica para la especificación de trabajar. Por lo tanto, esta especificación hace uno de dos supuestos:

- Sus Exteriores Puntos clave a otro campo / campos dentro de la corriente *JSON* tabla de esquema. En este caso se utiliza se emplea una palabra clave especial auto.

- Sus Exteriores Puntos clave a un campo / campos de una tabla de esquema *JSON* "en otro lugar". En este caso, la tabla de esquema *JSON* debe estar dentro de un recurso en el paquete de datos. Hay dos situaciones: o bien esta tabla de esquema *JSON* ya se encuentra dentro de un (tabular) El paquete de datos y la referencia es a un recurso dentro de este paquete de datos; O estamos señalando a un paquete (tabular) Los datos almacenados en otro lugar, por ejemplo, en línea.

Una clave externa es una referencia donde las entradas en un determinado campo (o campos) en esta tabla ("recurso" en la terminología de paquete de datos) es una referencia a una entrada en un campo (o campos) en un recurso independiente.

La propiedad ForeignKeys, si está presente, debe ser una matriz. Cada entrada de la matriz debe ser un ForeignKey. ForeignKey DEBE ser un hash y:

- Debe tener un campo de propiedad. campos es una cadena o una tabla que especifica el campo o campos de este recurso que forman la parte fuente de la clave externa. La estructura de la cadena o matriz es como por primaryKey anteriormente.

- Deben tener una referencia de la propiedad que debe ser un hash. el Hash

- Puede tener un datapackage propiedad. Esta propiedad es una cadena que apunta ser una ruta

de un paquete de datos o es el nombre de una datapackage. Si está ausente la implicación es que se trata de una referencia a un recurso dentro del paquete de datos actual o se trata de autorreferencia clave externa.

- Debe tener un recurso propiedad, que es el nombre del recurso dentro del paquete de datos de referencia. Para las claves externas de autorreferencia, el valor de los recursos debe ser auto.

- Debe tener un campos de propiedades que es una cadena si los campos exteriores es una cadena, de lo contrario una serie de la misma longitud que los campos exteriores, que describe el campo (o campos) referencias sobre el recurso de destino. La estructura de la cadena o matriz es como por primaryKey anteriormente.

He aquí un ejemplo:

```
"campos": [
 {
  "Name": "Estado"
 }
],
"llaves extranjeras": [
 {
  "campos": "Estado"
  "Referencia": {
   "Datapackage": "http://data.okfn.org/data/mydatapackage/",
   "Recurso": "el recurso",
   "campos": "state_id"
  }
 }
]
```

Un ejemplo de una clave externa que se autorreferencia:

```
"campos": [
 {
  "Name": "padre"
 },
 {
  "Name": "id"
 }
],
"llaves extranjeras": [
 {
  "campos": "padre"
  "Referencia": {
   "Datapackage": "",
   "Recurso": "sí",
   "campos": "id"
  }
 }
]
```

Tipos:

- *string* - UTF-8 cadena codificada hasta 64 KB de datos (a diferencia de 64K caracteres).
- *entero* - IEEE enteros con signo de 64 bits: [-263-1, 263-1]
- *float* - IEEE 754-2008 valores de coma flotante con formato
- *boolean* - "verdadero" o "falso", entre mayúsculas y minúsculas
- registro (*JSON* solamente) - un objeto *JSON*; también conocido como un registro anidado

E**squema XML**

Ver http://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes

- string
- booleano
- decimal
- float
- doble
- duración
- fecha y hora
- hora
- fecha
- gYearMonth
- gYear
- gMonthDay
- Gday
- gMonth
- hexBinary
- base64Binary
- anyURI